ESTIMATION DE COMPLEXITÉ ET LOCALISATION DE VÉHICULES À L'AIDE DE L'APPRENTISSAGE PROFOND

par

Frédéric Branchaud-Charron

Mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 18 janvier 2019

Le 18 janvier 2019

le jury a accepté le mémoire de Monsieur Frédéric Branchaud-Charron dans sa version finale.

Membres du jury

Professeur Pierre-Marc Jodoin
Directeur de recherche
Département d'informatique

Professeur Christian Desrosiers
Évaluateur externe
Département de Génie logiciel et des TI de l'ÉTS

Professeur Maxime Descoteaux
Président-rapporteur
Département d'informatique

Sommaire

L'analyse de la circulation routière est un domaine du génie civil permettant d'optimiser le déplacement des véhicules sur un système routier. Une étape importante de tout système d'analyse de trafic routier est la localisation des véhicules. Cette étape est effectuée à l'aide d'algorithmes d'apprentissage automatique, les réseaux de neurones à convolution.

Ce mémoire présente deux nouvelles bases de données de localisation et classification de véhicules permettant l'évaluation de techniques d'apprentissage modernes. Celles-ci contiennent plus de 648 959 véhicules classifiés parmi 11 classes. Un atout majeur de la base de données de localisation est la très grande variété de scènes permettant une meilleure évaluation des techniques dans plusieurs contextes différents.

Par la suite, on présente une technique d'estimation de la complexité d'une base de données. Cette technique permet d'analyser une base de données en un temps raisonnable et en apprendre plus sur sa composition. Elle permet d'estimer les performances atteignables par un algorithme d'apprentissage automatique sur cette base de données et d'en apprendre plus sur les relations entre les classes.

Finalement, une ébauche d'article sur une technique d'apprentissage automatique pour l'estimation d'orientation des véhicules est présentée en annexe. Cette méthode propose l'ajout d'un composant permettant l'«apprentissage en ligne» du modèle ce qui permet d'adapter le modèle à la scène proposée. Malgré cet ajout, ce modèle reste fiable pour la localisation et la classification tout en gardant sa rapidité d'exécution.

Mots-clés: apprentissage machine ; apprentissage profond ; localisation ; mesures de complexité.

Remerciements

Je tiens à remercier mon directeur de recherche, Pierre-Marc Jodoin, pour ses idées et son soutien tout au long de ma maitrise. Je le remercie de m'avoir fait participer au *workshop* TSWC pendant la conférence internationale CVPR 2017 ainsi qu'au Symposium AI de Montréal 2017. Je remercie le laboratoire VITAL (Clément, Mohammad, Martin, Yi, Zhimming, Francis, Philippe, Carl, Faezeh, Antoine, Jon, Charles et Thierry) pour les discussions intéressantes et les bons moments au laboratoire tout au long de mon parcours. Je remercie la compagnie Miovision (Andrew, Justin, Nick, Tyler, Dave, Joel, Mohan, Nada et Akshaya) pour leur soutien, le partage d'idées avant et pendant ma maitrise. Je remercie le FRQNT ainsi que MITACS pour leur soutien financier. Finalement, je remercie ma famille et ma conjointe pour leur soutien durant ces deux années.

Abréviations

iid indépendants identiquement distribués

TP *True Positive*

FP *False Positive*

TN *True Negative*

TP *False Negative*

SSD *Single-shot detector*

CNN *Convolutional neural network*

Table des matières

Liste des figures

Liste des tableaux

Introduction

L'explosion de l'accessibilité aux données et la disponibilité de meilleurs processeurs graphiques ont permis à l'apprentissage machine de connaître une révolution dans les dernières années. Par exemple, l'analyse de trafic routier a été grandement affectée par les réseaux de neurones à convolution qui ont permis d'améliorer les systèmes de reconnaissance de véhicules maintenant utilisés dans ce domaine.

De plus, nous observons une tendance forte vers la conception guidée par les données. Dans ce paradigme, on conceptualise l'algorithme d'apprentissage en fonction des caractéristiques des données que nous voulons modéliser. Ce domaine a subi un regain d'intérêts en raison du grand nombre d'applications et de base de données qui ont émergé suite à l'arrivée de l'apprentissage profond. Le domaine de l'analyse de complexité nous permet de mieux connaître les caractéristiques des bases de données.

0.1 Analyse de trafic routier

L'analyse de trafic routier permet aux divers paliers gouvernementaux de mieux comprendre la nature des déplacements sur leur réseau. Ainsi, ils peuvent mieux réguler les feux de circulation, mettre en place des voies réservées ou encore estimer l'usure de la route. Pour ce faire, les autorités doivent compiler des statistiques sur leurs routes telles que la fréquence des passages, le nombre de véhicules lourds ou encore analyser les embranchements à une intersection. L'étape initiale pour toutes ces tâches est la détection de véhicules. Plusieurs méthodes sont disponibles dont les boucles à inductions, les détecteurs infrarouges ou encore les systèmes de caméras[2]. Ces derniers sont particulièrement prisés en raison de leur flexibilité et leur facilité d'utilisation[19]. Malheureusement, les algorithmes modernes de détection de véhi-

cules utilisent le mouvement dans l'image, ce qui est inaccessible pour plusieurs systèmes de caméras[49]. En effet, bon nombre de caméras positionnées en bordure des routes enregistrent des vidéos à basse résolution et à faible nombre d'images par seconde, ce qui rend inutilisables les algorithmes se basant sur le mouvement tel que le flux optique.

Malgré l'importance de ce domaine pour les autorités, deux grands problèmes subsistent toujours : l'automatisation de la reconnaissance de véhicules dans un contexte réel et la disponibilité de bases de données de grande dimension dans ce domaine. Bien qu'il existe des algorithmes de localisation très performants notamment grâce à l'apprentissage profond[31, 87, 84, 63], ceux-ci sont entraînés sur des bases de données composées d'images non représentatives de scènes réelles. La qualité des images est souvent supérieure à celle obtenue dans un contexte d'application réelle, souvent corrompues par des artefacts de compression et leur basses résolutions. De plus, il y a peu de bases de données dédiées à l'entraînement et la validation d'algorithmes de localisation de véhicules. Ces bases de données[109, 30, 17, 114] sont souvent peu complexes, peu diversifiées et contiennent peu d'exemples. C'est pourquoi le défi MioTCD[66] a été mis sur pied à l'occasion de la conférence internationale CVPR 2017. Ce défi présentait une nouvelle base de données de véhicules annotée par plus de 200 personnes avec plus de 500,000 images dédiées à la classification et 100,000 images dédiées à la localisation. En plus de sa taille, celle-ci contient une grande variété de véhicules, de saisons et de contextes. Dans le contexte de ce projet, j'ai testé plusieurs méthodes de classification et de localisation. De plus, on aimerait pouvoir avoir une estimation de la direction du véhicule pour améliorer les algorithmes de traque. Cette estimation est difficile sans information sur le mouvement dans la scène. En utilisant des réseaux à convolution, j'ai développé un modèle pouvant faire cette estimation.

0.2 Analyse de complexité

Suite au défi MioTCD, les organisateurs ont été étonnés des impressionnantes performances des algorithmes d'apprentissage profond, notamment en classification. En effet, même les réseaux de neurones les plus simples obtenaient des scores approchant la perfection. Pourtant, malgré toutes les avancées en apprentissage automatique,

plusieurs jeux de données comportent des caractéristiques les rendant difficiles, voire impossibles à décrire par les algorithmes modernes. À l'inverse, certaines banques de données sont d'une simplicité déconcertante. On aimerait prendre avantage de cette information lors de la sélection de notre algorithme. C'est le but de l'analyse de complexité qui extrait des caractéristiques telles que la confusion entre classes, la difficulté d'apprentissage ou encore les caractéristiques discriminantes. Dans ce domaine, un jeu de données est composé de points dans un espace $\mathbb{R}^d$ où d est le nombre de dimensions (ou caractéristiques) du problème. On cherche à caractériser les relations entre ces points ou leurs étiquettes respective. Il est important de noter que l'ensemble de données ne représente pas la distribution complète du problème, mais bien une partie. Ces échantillons peuvent être non représentatifs de la vraie distribution ce qui complique l'apprentissage et la généralisation du modèle.

L'analyse de complexité a été utilisée dans plusieurs domaines tels que le débruitage[92], la sélection de classificateurs[10], la sélection de caractéristiques[80], la sélection des hyperparamètres[70] et la sélection d'instances[59]. Il est possible de décrire les relations entre les classes, car bien que souvent non linéairement séparables, elles possèdent une structure propre ainsi que des caractéristiques permettant de les différencier. Par exemple, dans leur espace à très haute dimension, les images de chats sont regroupées et occupent un espace différent des images d'automobiles.

Il existe quatre sources de complexités[41] : i) l'ambiguïté, ii) la complexité des frontières entre classes, iii) la taille de l'échantillon par classe et iv) le nombre de dimensions. L'**ambiguïté** entre les classes survient lorsqu'il est impossible d'assigner une classe à un échantillon. Cela peut être la conséquence de la nature du problème, si les deux classes représentent les mêmes concepts ou alors, il n'existe pas de caractéristique discriminante pour les distinguer. Dans ce cas, il n'y a d'autres choix que d'améliorer la base de données en modifiant le problème ou en ajoutant de nouvelles caractéristiques. Par exemple si l'on a les classes chat et chien, on ne peut pas les distinguer si l'on ne possède que les caractéristiques : couleur et nombre de pattes. C'est ce que l'on appelle des caractéristiques non discriminantes. Par contre, si l'on ajoute l'attribut type de cri, on peut facilement différencier les chats qui miaulent des chiens qui jappent.

La **complexité des frontières** devient un problème lorsque la séparation entre

les classes est très complexe. Cela complique la tâche du classificateur, car il doit estimer une fonction de séparation plus complexe.

La **taille de la base de données** et le **nombre de dimensions** contribuent tous les deux au nombre de régions clairsemées dans l'espace et peuvent entraîner une mauvaise généralisation du modèle. Cette pathologie est nommée la malédiction de la dimensionnalité. Pour atténué ce problème, on doit ajouter des données à l'ensemble d'entraînement ou encore, projeter les données vers un autre espace dimensionnel comme le font les réseaux de neurones. Il est possible de le faire artificiellement en augmentant la quantité de données déjà présentes. Par exemple, on peut ajouter du bruit ou modifier nos images pour créer de nouveaux exemples.

0.3 Plan du mémoire

Le premier chapitre présente les bases de l'apprentissage machine et de la vision par ordinateur. Le deuxième présente les mesures de complexités. Le troisième chapitre présente notre publication soumise à la conférence internationale CVPR 2019 qui propose une nouvelle mesure de complexité basée sur la théorie des graphes. Notre publication sur la base de données MioTCD est disponible en annexe A. L'annexe B présente une ébauche de publication en vue d'une soumission au journal T-ITS sur une nouvelle application de l'apprentissage profond pour estimer l'orientation des véhicules photographiés par des caméras de surveillance.

Chapitre 1

Algorithmes d'apprentissage

Les algorithmes d'apprentissages font partie intégrante de l'intelligence artificielle. Contrairement aux méthodes aux règles préétablies, ceux-ci extraient des règles à partir des données qui leur sont fournies. Pour ce faire, ils utilisent des méthodes issues des domaines de l'optimisation, des statistiques et de la théorie de l'information.

1.1 Probabilités

Les probabilités sont utilisées en apprentissage automatique pour caractériser l'incertitude des données. En effet, celles-ci sont souvent extraites d'environnements affectés par des événements aléatoires.

Une **variable aléatoire** est une variable pouvant prendre plusieurs états. La variable est issue d'une distribution qui donne la probabilité d'obtenir chacun des états.

Bien qu'une distribution peut être discrète ou continue, dans ce mémoire nous nous concentrons sur le cas discret. Dans le cas discret, on peut décrire une distribution par une fonction de masse indiquant la probabilité qu'une variable prenne un état. Soit une variable X et un état x, la probabilité d'un événement $X = x$ est notée $P(X = x)$. Une distribution doit posséder les propriétés suivantes :

— Tous les états de la variable aléatoire sont représentés.

— Un état impossible obtient la probabilité 0.

— $\sum_{x \in X} P(X = x) = 1$, la somme de la probabilité de tous les événements est 1.

Pour simplifier la notation, on représentera $P(X = x)$ par $P(x)$. On désigne la probabilité marginale comme étant la probabilité d'un sous-ensemble de variables. Soient X, Y deux variables aléatoires iid de $P(x, y)$, on peut représenter la probabilité d'une variable en marginalisant (sommant) les autres : $P(x) = \sum_{y \in Y} P(x, y)$. La règle est analogue pour le cas continu où l'on utilise une intégrale.

Pour désigner la probabilité d'un événement étant donné une observation, on calcule la probabilité jointe divisée par la probabilité de l'observation :

$$P(y|x) = \frac{P(y, x)}{P(x)}.$$

À noter que cette valeur est valide seulement pour $P(x) > 0$. On nomme cette probabilité la probabilité conditionnelle.

Lorsque deux variables sont indépendantes, la probabilité jointe peut être exprimée comme un produit de probabilités :

$$\forall x \in X, y \in Y, P(x, y) = P(x)P(y)$$

ou encore que

$$P(x \mid y) = P(x),$$
$$P(y \mid x) = P(y).$$

Deux variables aléatoires X, Y sont conditionnellement indépendantes à une variable aléatoire Z, lorsqu'il est possible d'exprimer la probabilité jointe comme un produit de deux probabilités marginales. On a donc $\forall x \in X, y \in Y, z \in Z$:

$$P(x, y|z) = P(x|z)P(y|z).$$

Lorsqu'on connaît $P(Y|X)$, $P(X)$ et que l'on veut connaître $P(X|Y)$, on peut

utiliser la règle de Bayes qui se formule ainsi :

$$P(X|Y) = \frac{P(X)P(Y|X)}{P(Y)}$$
$$P(Y) = \sum_{x \in X} P(Y|x)P(x).$$

1.1.1 Espérance mathématique

L'espérance mathématique d'une fonction f par rapport à une distribution $P(x)$ est la valeur moyenne de f lorsque X est issue de P. Pour le cas discret, on a :

$$E_{X \sim P}[f(x)] = \sum_{x \in X} P(x)f(x).$$

1.1.2 Distribution gaussienne

En apprentissage machine, et dans bien d'autres domaines, on utilise fréquemment la loi gaussienne (ou normale) pour approximer les distributions aléatoires qu'on dénote $\mathcal{N}(\mu, \sigma^2)$.

Si la variable aléatoire X suit une loi normale : $X \sim \mathcal{N}(\mu, \sigma^2)$, le paramètre μ dénote la moyenne de la distribution c'est-à-dire $E[X] = \mu$ et la variance de la distribution est donnée par σ^2. On utilise la loi normale pour plusieurs raisons. Tout d'abord, le théorème central limite établit que la somme de plusieurs variables aléatoires tend vers une distribution normale. Deuxièmement, lorsqu'on ne connaît pas la vraie distribution d'un problème, la loi normale est celle qui impose le moins de contraintes[88]. Il est souvent difficile, voire impossible de connaître la vraie distribution du problème que l'on tente de résoudre, c'est pourquoi la loi normale est fréquemment utilisée.

On utilisera le vecteur $\vec{\mu}$ et la matrice de covariance Σ pour représenter une distribution à plusieurs dimensions. En pratique, on aura besoin de calculer la densité d'une distribution gaussienne comme suit :

$$\mathcal{N}(\vec{x} \mid \vec{\mu}; \Sigma) = \frac{1}{(2\pi)^{d/2}} \frac{1}{|\Sigma|^{1/2}} exp(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \Sigma^{-1}(\vec{x} - \vec{\mu})) \tag{1.1}$$

où d est le nombre de dimensions de la gaussienne.

1.2 Théorie de l'information

Le but de la théorie de l'information est de quantifier la quantité "d'information" contenu dans un événement x. En ce sens, plus un événement est rare, plus ce dernier est riche en information.

L'information d'un événement x dans le cas discret est donnée par :

$$I(x) = -\log(P(x))$$

où l'on quantifie cette valeur comme un *bit* si on utilise $\log_2$ et un *nat* si l'on utilise le logarithme naturel. On peut définir l'incertitude d'une distribution avec l'entropie de Shannon : $H(X) = \mathbb{E}_{X\sim P}[I(x)]$ qui exprime la quantité de bits moyen pour encoder un événement $x \in X$.

Il est possible de comparer deux distributions grâce à la théorie de l'information. On note la divergence entre deux distributions la divergence de Kullback-Leibler (KL), qu'on dénote $D_{KL}(P \parallel Q)$ où $P(X)$ la distribution cible et $Q(X)$ son approximation. Cette divergence permet d'exprimer le nombre de bits exédentaires requis pour encoder une variable $X \sim P(X)$ alors qu'on utilise la distribution $Q(X)$:

$$D_{KL}(P \parallel Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}.$$

La D_{KL} a deux propriétés fondamentales :

— $D_{KL}(P \parallel Q) = 0 \rightarrow P = Q$.

— Elle n'est pas symétrique et toujours positive, $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$.

En pratique, on utilise l'entropie croisée $H(P, Q)$, car elle est plus rapide à calculer tout en donnant le même résultat. Soit l'entropie croisée :

$$H(P, Q) = -\sum_x P(x) \log Q(x).$$

On peut exprimer l'entropie croisée en fonction de la KL divergence :

$$\begin{aligned} D_{KL}(P \parallel Q) &= \sum_x P(x) \log \frac{P(x)}{Q(x)} \\ &= \sum_x -P(x) \log Q(x) + P(X) \log Q(x) \\ &= -\sum_x P(x) \log Q(x) + \sum_x P(X) \log P(x) \\ &= -\sum_x P(x) \log Q(x) - \sum_x P(X) \log \frac{1}{P(x)} \\ &= -\sum_x P(x) \log Q(x) - H(P) \\ &= H(P,Q) - H(P) \\ H(P,Q) &= H(P) + D_{KL}(P \parallel Q). \end{aligned}$$

En apprentissage automatique, $P(X)$ représente la distribution des données. C'est donc une distribution fixe. On a donc :

$$H(P,Q) \simeq D_{KL}(P \parallel Q).$$

1.3 Apprentissage machine

On peut définir un algorithme d'apprentissage machine comme un programme informatique capable d'apprendre à partir d'**expériences** pour améliorer sa **performance** sur une **tâche** donnée[74]. Plus formellement, c'est un algorithme qui entraîne un modèle f aux paramètres $\vec{w}$ sur un jeu de données $\mathcal{D} = (\vec{x}_0, y_0), \ldots, (\vec{x}_n, y_n)$ pour minimiser une fonction de perte $\mathcal{L}$. Soit $\vec{x}_i \in \mathbb{R}^d$ une donnée observée et y_i sa cible. Tel que mentionné à la Section 1.3.5, l'apprentissage machine s'attarde principalement sur deux tâches : la classification et la régression. En classification, on aura un algorithme d'apprentissage qui doit correctement assigner une étiquette de classe $y_i \in \{c_1, c_2, \ldots, c_K\}$ à la donnée observée. En régression, on aura $y_i \in R^l$ une cible de valeur numérique. Le but d'un modèle $f_{\vec{w}}(\vec{x}_i)$ est de retrouver la cible y_i pour toute donnée $\vec{x}_i$.

1.3.1 Les expériences

Les algorithmes d'apprentissage automatiques peuvent être généralement catégorisés en deux catégories : les algorithmes supervisés et non supervisés. Cette distinction exprime la manière dont les algorithmes utilisent les données lors de l'entraînement.

Algorithmes supervisés

Les algorithmes supervisés tentent de prédire une cible y_i en fonction de l'entrée $\vec{x}_i$. Pour ce faire, chaque exemple est accompagné d'une cible et le modèle doit prédire celle-ci. Ainsi, le modèle est guidé dans son apprentissage par les cibles qui lui sont proposées. Par exemple, la classification est un exemple d'algorithme supervisé.

Algorithmes non supervisés

Les algorithmes non supervisés n'ont pas besoin d'étiquette pour l'entraînement, ils tirent avantage de la structure des données et des regroupements à l'intérieur de l'ensemble de données. Les algorithmes de débruitage ou de regroupement font partie de cette catégorie. Par exemple, sans avoir les étiquettes, on pourrait segmenter les pixels bleus d'une image automatiquement pour trouver le ciel.

1.3.2 Maximum de vraisemblance conditionnel

L'apprentissage machine cherche à obtenir un modèle $f_{\vec{w}}(\vec{x}_i)$ qui maximise $P(Y \mid X; \vec{w})$, soit la probabilité des étiquettes $Y = \{\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_n\}$ étant donné les données $X = \{\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_n\}$ et les paramètres $\vec{w}$ du modèle :

$$\hat{w} = \arg\max_{\vec{w}} P(Y \mid X; \vec{w}).$$

En assumant l'indépendance des données, on obtient :

$$\hat{w} = \arg\max_{\vec{w}} \prod_{i=1}^{n} P(\vec{y}_i \mid \vec{x}_i; \vec{w}).$$

Cette notation peut être prohibitive notamment pour les débordements inférieurs en raison du produit des probabilités. En utilisant le logarithme de la fonction, on

peut reformuler la fonction précédente par une somme :

$$\hat{w} = \arg\max_{\vec{w}} \sum_{i=1}^{n} \log P(y_i \mid \vec{x}_i; \vec{w}).$$

Lorsque $n \to \infty$, le maximum de vraisemblance estime un $\hat{w}$ qui converge vers la valeur idéale w^*.

1.3.3 Maximum *a posteriori*

Lorsqu'on dispose d'un *a priori* sur les paramètres $\vec{w}$, il est possible d'utiliser cette connaissance avec l'approche bayésienne du Maximum *a posteriori*.

Dans ce cas, on veut trouver les paramètres $\vec{w}$ les plus probables étant donné l'entrée X et les observations Y :

$$\begin{aligned}
\hat{w} &= \arg\max_{\vec{w}} P(\vec{w} \mid X, Y) \\
&= \arg\max_{\vec{w}} \frac{P(Y \mid \vec{w}; X) P(\vec{w}; X)}{P(Y)} \\
&\propto \arg\max_{\vec{w}} \prod_{(\vec{x}_i, \vec{y}_i) \in \mathcal{D}} P(\vec{y}_i \mid \vec{x}_i; \vec{w}) P(\vec{w}).
\end{aligned}$$

L'utilisation du MAP permet d'optimiser deux critères indépendamment : le terme de vraisemblance et le terme *a priori* :

$$\begin{aligned}
\hat{w} &= \arg\max_{\vec{w}} \log \prod_{(\vec{x}_i, \vec{y}_i) \in \mathcal{D}} P(\vec{y}_i \mid \vec{x}_i; \vec{w}) P(\vec{w}) \\
&= \arg\max_{\vec{w}} \log \prod_{(\vec{x}_i, \vec{y}_i) \in \mathcal{D}} P(\vec{y}_i \mid \vec{x}_i; \vec{w}) + \log P(\vec{w})
\end{aligned}$$

En assumant l'indépendance des variables aléatoires :

$$= \arg\max_{\vec{w}} \sum_{i=1}^{n} \log P(y_i \mid \vec{x}_i; \vec{w}) + \log P(\vec{w}).$$

1.3.4 La performance

Pour évaluer les algorithmes d'apprentissage, on utilise une mesure de performance ou fonction perte (*loss* en anglais). Cette mesure est généralement élevée si le modèle fait une mauvaise prédiction et faible sinon. Plus formellement, soit un modèle $f_{\vec{w}}(\vec{x}_i)$, sa cible à prédire $y_i \in R^k$ et sa fonction de perte $\mathcal{L} : R^k \times R^k \to R$. On choisira le modèle qui minimisera le risque empirique :

$$\arg\min_{\vec{w}} \mathbb{E}[L] = \arg\min_{\vec{w}} \frac{1}{n} \sum_{i}^{n} \mathcal{L}(f_{\vec{w}}(\vec{x}_i), y_i).$$

En classification, on utilise souvent l'entropie croisée vue plus haut pour pénaliser les distributions divergentes. À l'opposé, en régression l'erreur quadratique moyenne (MSE) est utilisée :

$$MSE(\vec{x}, \vec{y}) = \frac{1}{k} \sum_{i=1}^{d} (\vec{x}_i - \vec{y}_i)^2. \tag{1.2}$$

Dans certains cas, certaines connaissances a priori permettent de choisir une meilleure fonction de perte. Par exemple en segmentation cardiaque, il est impossible pour les ventricules de ne pas être à l'intérieur du myocarde. On pourrait utiliser cette connaissance pour pénaliser les prédictions qui vont à l'encontre de cette information.

1.3.5 La tâche

Il existe deux familles de tâches particulièrement prisées en apprentissage machine : la classification et la régression. Plusieurs sous-tâches existent telles que la détection d'anomalies, la synthèse d'exemples, le débruitage et plus encore. La tâche est intimement liée à la composition des cibles y_i de l'ensemble $\mathcal{D}$.

Classification

Un algorithme de classification doit décider à quelle catégorie appartient un échantillon $\vec{x}$ parmi K catégories. Plus formellement, soit $\vec{x} \in R^d$ et $y \in \{c_1, c_2, \ldots, c_K\}$, on a que $y = f(\vec{x}), f : R^d \to \{c_1, c_2, \ldots, c_K\}$. Lorsque $K > 1$, il est possible que f produise une distribution qui décrit la probabilité que $\vec{x}_i$ appartienne à chacune des classes. On prend alors la classe maximale de celle-ci soit la classe avec la plus grande confiance. La classification est utilisée notamment pour la reconnaissance d'images. Par exemple, ImageNet[23] est un jeu de données où il faut catégoriser un objet parmi 1000 classes.

Pour le cas à deux classes, où $y_i \in \{0, 1\}$, on aura la probabilité postérieure suivante qui est dérivée de la distribution de Bernouilli :

$$P(y_i \mid f_{\vec{w}}(\vec{x}_i)) = f_{\vec{w}}(\vec{x}_i)^{y_i}(1 - f_{\vec{w}}(\vec{x}_i))^{1-y_i}.$$

Toujours en supposant l'indépendance des exemples, on peut alors trouver $\vec{w}$ suivant un maximum de vraisemblance :

$$\begin{aligned}
\vec{w} &= \arg\max_{\vec{w}} \log(P(Y \mid X; \vec{w})) \\
&= \arg\max_{\vec{w}} \log \prod_{i}^{n} P(y_i \mid f_{\vec{w}}(\vec{x}_i)) \\
&= \arg\min_{\vec{w}} -\sum_{i}^{n} y_i f_{\vec{w}}(\vec{x}_i) + (1 - y_i) \log(1 - f(\vec{x}_i)).
\end{aligned}$$

Régression

Dans le cas de la régression, un algorithme prédit une valeur numérique et non une étiquette de classe. Plus formellement, pour un couple $\vec{x}_i \in \mathbb{R}^d$ et $\vec{y}_i \in \mathbb{R}^l$, on souhaite que $\vec{y}_i \simeq f_{\vec{w}}(\vec{x}), f : \mathbb{R}^d \to \mathbb{R}^l$. Un exemple serait un modèle prédisant les prochaines valeurs boursières à partir des précédentes journées.

Si on suppose que les données sont affectées par un bruit gaussien, on voudra estimer la moyenne de cette gaussienne pour retrouver les cibles originales.

Pour des données iid, on obtient la distribution de vraisemblance suivante :

$$P(Y \mid X; \vec{w}; \Sigma) = \prod_{i}^{n} \mathcal{N}(\vec{y} \mid f_{\vec{w}}(\vec{x}_i); \Sigma).$$

À l'aide de l'équation de densité (Eq. 1.1), on peut retrouver les meilleurs paramètres $\vec{w}$ à l'aide du maximum de vraisemblance :

$$\begin{aligned}\vec{w} &= \arg\min_{\vec{w}} -\sum_{i}^{n} \log \mathcal{N}(\vec{y}_i \mid f_{\vec{w}}(\vec{x}); \Sigma) \\ &= \arg\min_{\vec{w}} -\sum_{i}^{n} \log \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} - \sum_{i}^{n}(-\frac{1}{2}(\vec{y}_i - f_{\vec{w}}(\vec{x}))^T \Sigma^{-1}(\vec{y}_i - f_{\vec{w}}(\vec{x}))).\end{aligned}$$

Si on retire les termes qui ne dépendent pas de $\vec{w}$ et qu'on suppose l'indépendance des variables, on peut retirer Σ :

$$\begin{aligned}&= \arg\min_{\vec{w}} \sum_{i}^{n}(\frac{1}{2}(\vec{y}_i - f_{\vec{w}}(\vec{x}_i))^T(\vec{y}_i - f_{\vec{w}}(\vec{x}_i))) \\ &= \arg\min_{\vec{w}} \sum_{i}^{n} \frac{1}{2}(\vec{y}_i - f_{\vec{w}}(\vec{x}_i))^2.\end{aligned}$$

On remarque qu'on obtient une fonction de perte comparable à la MSE(Eq. 1.2) vue précédemment.

Localisation

La localisation combine la classification et la régression. La tâche consiste à placer des boîtes autour des objets présents dans la scène et de les classifier. En général, les boîtes sont décrites par leurs extrêmes en x, y. Cette tâche est particulièrement difficile dû à la grande variété d'objets à trouver. Par exemple, COCO[61] comporte près de 200 classes d'objets différents.

1.3.6 Entraînement

Tel qu'indiqué plus haut, l'entraînement d'un algorithme d'apprentissage consiste à trouver les paramètres $\vec{w}$ d'un modèle $f_{\vec{w}}(\vec{x})$ qui minimiseront la fonction de perte $\mathcal{L}$ sur l'ensemble d'entraînement $\mathcal{D}$. Bien que des solutions fermées existent, elles sont généralement impraticables en analyse d'images dû à la complexité du problème et au grand nombre de données. En effet, ce type d'approches impliquent le calcul de l'inverse d'une matrice $N \times N$, qui est impossible à calculer lorsque N est grand. On privilégiera explorer l'espace de solutions de façon itérative jusqu'à convergence en optimisant les paramètres $\vec{w}$ au fur et à mesure de l'entraînement.

Optimisation par descente de gradient

La première solution est de calculer le gradient de l'erreur pour chaque couple $(\vec{x}_i, \vec{y}_i) \in \mathcal{D}$ et d'en faire la moyenne. Le gradient pointera alors dans la direction où $\mathcal{L}$ grandira le plus. On se déplace donc dans la direction opposée pour diminuer la fonction de perte :

$$\vec{w}^{t+1} = \vec{w}^t - \alpha(\nabla_{\vec{w}^t}\mathcal{L}(\mathcal{D}, \vec{w}^t)) \tag{1.3}$$

$$= \vec{w}^t - \alpha(\frac{1}{N}\sum_{i=1}^{N}\nabla_{\vec{w}^t}\mathcal{L}(\mathcal{D}_i, \vec{w}^t)) \tag{1.4}$$

où α est le pas d'entraînement et détermine la vitesse d'apprentissage. Un α trop élevé ou trop bas peut nuire à la convergence de l'algorithme et l'empêcher d'atteindre la solution optimale. On met à jour les poids jusqu'à ce que la fonction de perte atteigne un seuil ou après un nombre d'itérations fixe.

Descente de gradient stochastique

On remarque que la méthode par descente de gradient peut être inappropriée pour de grands ensembles $\mathcal{D}$. En effet, lorsque N est grand, il est extrêmement coûteux de calculer les gradients pour l'ensemble. Par conséquant, on fera les mises à jour après chaque couple $(\vec{x}_i, y_i)$. Cela permet d'avancer plus rapidement dans l'espace des solutions. Malheureusement, cette technique rend l'exploration de l'espace de

solutions très aléatoire tout en étant lente à converger.

Input: Un ensemble $\mathcal{D}$.
Une fonction de perte $\mathcal{L}$.
Un modèle f aux paramètres $\vec{w}$.
Un critère d'arrêt $Critère()$.
Nombre d'exemples par lot b.
while *!Critère()* **do**
  $B \leftarrow b$ exemples tirés iid de $\mathcal{D}$;
  for $k = 0, 1, \ldots, b$ **do**
    $g_k \leftarrow \nabla_{\vec{w}} \mathcal{L}(_k, \vec{w})$;
  end
  $g \leftarrow \frac{1}{b} \sum_k^b g_k$;
  $\vec{w} = \vec{w} - \alpha(g)$;
end

Algorithm 1: Algorithme de descente de gradient par lot.

Une façon d'accélérer le processus et d'être plus résiliant au bruit est de faire de l'entraînement par lot. Contrairement à la descente de gradient stochastique qui utilise un seul exemple par mise à jour, on en utilisera plusieurs qu'on moyennera avant de faire la mise à jour des poids. Cela permet d'avancer plus rapidement dans l'espace de solutions. Cet algorithme est décrit par Alg. 1. Pour la descente de gradient stochastique classique, on choisira $b = 1$.

Un concept important est celui d'*epoch*, il désigne le nombre de fois que l'algorithme a itéré sur le jeu de données au complet. Par exemple, dans l'algorithme de descente de gradient, on ne fait qu'une mise à jour par *epoch*.

Sous apprentissage et sur apprentissage

Suite à l'entraînement, on aimerait que notre modèle performe bien sur de nouvelles données et ainsi être utilisé dans un contexte réel. C'est ce qu'on appelle un processus de **généralisation**.

Un problème fort bien documenté survient lorsque la capacité d'un modèle est trop élevée ce qui lui permet de «mémoriser» les données d'entraînement. Cela survient lorsque le modèle apprend une frontière de décision trop complexe qui ne pourra pas bien généraliser. On dira alors que le modèle souffre de sur apprentissage. On

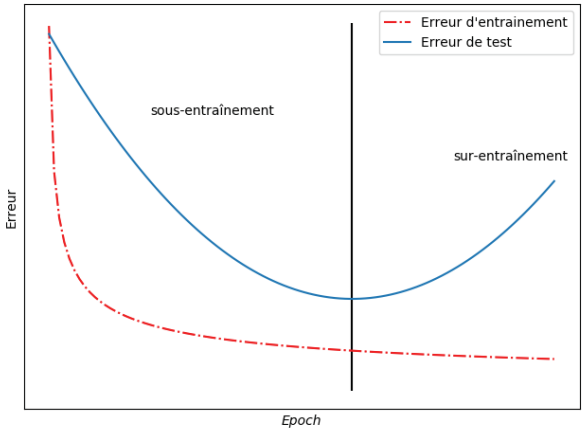


figure 1.1 – Illustration du déroulement d'un entraînement typique. Si l'ensemble de test est fiable, le modèle qui généralisera le mieux se trouve près de la ligne verticale.

peut diagnostiquer ce problème lorsque l'erreur d'entraînement est basse et l'erreur sur l'ensemble de test est élevée. On peut compenser ce problème en régularisant les paramètres du modèle pour l'empêcher d'apprendre une fonction de décision trop complexe.

À l'inverse, lorsque la frontière n'est pas assez fidèle aux données, le modèle souffrira de sous-entraînement, là où les erreurs d'entraînement et de test sont élevées. Dans ce cas, on peut prolonger l'entraînement, projeter les données dans un espace dimensionnel plus simple ou modifier le modèle (c.f., Fig. 1.1).

1.3.7 Métriques d'évaluation

Il est essentiel d'évaluer les modèles pour valider la qualité des prédictions sur de nouvelles données et de s'assurer de la bonne généralisation de notre modèle. Pour ce faire, on sépare notre ensemble $\mathcal{D}$ pour obtenir un ensemble de test sur lequel le modèle ne s'entraînera pas. Suite à l'entraînement, on analyse sa capacité de généralisation grâce aux prédictions sur l'ensemble de test.

Les métriques qu'on utilise dépendent du problème à résoudre. En effet, une connaissance du domaine peut nous indiquer s'il y a des coûts différents quant aux erreurs de prédiction du modèle. Par exemple, un système d'alarme qui génère quelques faux positifs peut être sans conséquence, mais s'il génère des faux négatifs cela peut

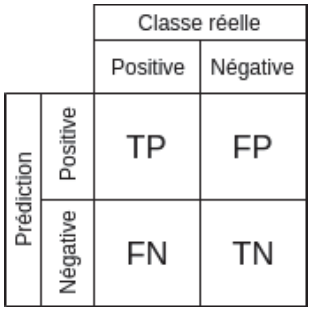


figure 1.2 – Calcul des attributs pour le cas deux classes.

être coûteux. Il est important de noter qu'aucune métrique n'est parfaite et peut être utilisée à tort.

Classification

Pour évaluer les algorithmes de classification 2 classes, on combine les attributs suivants : vrai positif (TP), faux positif (FP), vrai négatif (TN) et faux négatif (FN). On peut calculer ces attributs avec une matrice de confusion celle de la Fig. 1.2.

La première mesure est la justesse (*accuracy* en anglais), elle dénote le ratio d'exemples correctement classifiés sur le nombre total d'exemples. Cette mesure n'est pas recommandée si les classes sont fortement déséquilibrées, car les erreurs des classes moins nombreuses auront trop peu d'influence :

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

Pour nuancer notre analyse, on fait appel à la précision et au rappel. La précision est élevée lorsque le nombre de faux positifs est faible et le nombre de vrais positifs est faible. À l'opposé, le rappel est élevé si le nombre de faux négatifs est faible et le nombre de vrais positifs est élevé. Ces deux métriques ont des objectifs très différents. Pour le rappel, il est important que tous les éléments de la classe positive soient identifiés quitte à générer un grand nombre de faux positifs. La précision accorde plus d'importance aux faux positifs, peu importe le nombre de faux négatifs :

$$Precision = \frac{TP}{TP + FP}$$
$$Rappel = \frac{TP}{TP + FN}.$$

Étant donné leurs objectifs diamétralement opposés, il est dangereux d'utiliser qu'une seule de ces deux métriques. On privilégiera alors la *F-measure*, qui est une combinaison de la précision et du rappel :

$$F_\beta = (1 + \beta^2) * \frac{Precision * Rappel}{\beta^2 Precision + Rappel}.$$

Le paramètre β permet de mettre l'accent sur la précision ou le rappel. En général on utilise $\beta = 1$.

Finalement, bien qu'on classifie généralement une donnée dans la classe positive si la probabilité est plus grande que 0.5, il est intéressant de faire varier ce seuil. En calculant la précision et le rappel à plusieurs niveaux de seuil, on obtient une courbe appelée la courbe précision-rappel. Lorsque le seuil est bas, on obtient un rappel élevé, mais plus de faux positifs. Lorsque le seuil est haut, on obtient peu de faux positifs, mais beaucoup de faux négatifs. On utilise cette courbe pour choisir le meilleur seuil selon notre application. Si on prend par exemple la Fig. 1.3, on y compare trois courbes précision-rappel obtenues de trois modèles différents. Généralement, on choisit le modèle le plus près du coin supérieur droit. Dans ce cas-ci, la courbe continue verte est choisie, car elle permet de garder une bonne précision tout en ayant un bon rappel.

Localisation

Il est extrêmement difficile d'évaluer les algorithmes de localisation due à la grande variété des problèmes dont peuvent souffrir les algorithmes. Par exemple, un algorithme peut produire des boîtes trop grandes, mais trouver beaucoup d'objets alors qu'un autre génère des boîtes parfaites en omettant les petits objets. De ce fait, l'ef-

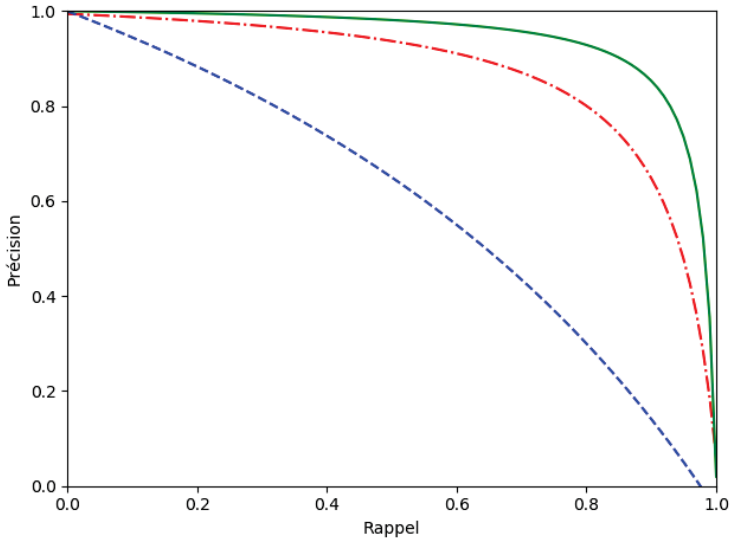


figure 1.3 – Exemple de trois courbes précision-rappel.

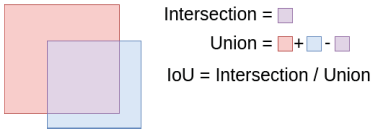


figure 1.4 – Exemple du calcul de l'IoU.

fort s'est concentré sur les protocoles d'évaluation pour les rendre plus nuancées et non sur les métriques.

La première mesure à utiliser est l' *intersection over union (IoU)*. Elle permet de comparer deux boîtes et savoir à quel point elles sont similaires. Comme dans la Fig. 1.4, on calcule l'aire où les deux boîtes se chevauchent, divisé par l'aire de l'union des deux boîtes. En général, on note qu'une $IoU > 0.5$ implique que les deux boîtes sont assez similaires.

La principale métrique est la *mean Average Precision* (mAP), elle est notamment utilisée pour les défis COCO[61] et Pascal VOC [27]. Soit un ensemble de boîtes cibles et de boîtes prédites. Pour chaque boîte cible dont le $IoU > 0.5$ avec une boîte prédite sera considérée comme un vrai positif, les autres étant considérées comme des faux positifs. En utilisant la confiance des boîtes, on peut générer plusieurs précisions pour plusieurs niveaux de rappel. On prendra la moyenne des précisions à plusieurs niveaux

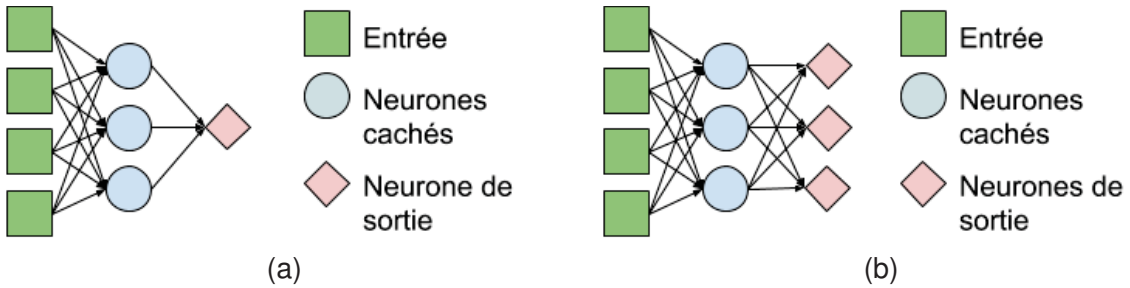


figure 1.5 – Représentation graphique d'un réseau de neurones. a) Exemple d'un réseau pour un problème binaire à 4 neurones. b) Exemple d'un réseau pour un problème trois classes à 6 neurones.

de rappel.

Cette métrique est fortement critiquée, elle encourage la création de boîtes à faible confiance pour obtenir des scores de rappel élevé. De plus, le seuil 0.5 pour l'IoU est fortement critiqué, car il avantage les techniques qui génèrent des boîtes imparfaites. Le processus d'évaluation de COCO[61] a mitigé cette contrainte en rapportant plusieurs seuils d'IoU.

1.4 Réseaux de neurones et apprentissage profond

L'apprentissage profond regroupe les algorithmes utilisant les réseaux de neurones pour effectuer leur tâche. On définit un neurone comme une fonction $h(\langle \vec{w}, \vec{x} \rangle + b)$ où $\vec{x}$ sont les données en entrée, $\vec{w}$ représente le vecteur de poids appris et b est le biais également appris. On applique une fonction non linéaire h au résultat pour que le neurone puisse approximer des fonctions non linéaires.

On représente souvent les réseaux de neurones sous la forme d'un graphe orienté où les neurones sont connectés par des arrêtes comme à la Fig. 1.5. Pour simplifier la notation, on inclut le biais b dans les poids $\vec{w}$ et ajoute une dimension au vecteur $\vec{x}$: $\vec{x}' = (\vec{x}, 1)$, $\vec{w}' = (\vec{w}, b)$.

1.4.1 Perceptron

Le perceptron [90] est le premier réseau de neurones jamais publié. Il est défini comme $y = h(\langle \vec{w}, \vec{x} \rangle)$ où h est la fonction de non-linéarité signe. Il est utilisé pour faire de la classification binaire :

$$h(\langle \vec{w}, \vec{x} \rangle) = y = \left\{ \begin{array}{ll} -1, & \text{si } \langle \vec{w}, \vec{x} \rangle < 0 \\ 1, & \text{sinon} \end{array} \right\}.$$

La fonction de perte $\mathcal{L}$ est particulière pour ce modèle, on l'appelle le critère du perceptron :

$$\mathcal{L}_{percep}(\mathcal{D}'; \vec{w}) = - \sum_{(\vec{x}_i, y_i) \in \mathcal{D}'} y_i (\vec{w} \vec{x}_i)$$

où $\mathcal{D}'$ est l'ensemble des exemples mal-classés. Les poids de ce modèle sont appris par descente de gradient. On a donc besoin de calculer le gradient de cette fonction.

$$\nabla_{\vec{w}} \mathcal{L}_{percep} = \sum_{(\vec{x}_i, y_i) \in \mathcal{D}'} y_i \vec{x}_i.$$

Malheureusement, le gradient est nul lorsque toutes les données sont bien classées et ce même si la frontière de décision apprise est collée aux données. De plus, on aimerait obtenir une distribution de probabilité sur les classes pour prendre notre décision. Pour régler ces problèmes, on utilise la **régression logistique**. Au lieu d'estimer la classe y_i de l'entrée $\vec{x}_i$, on calculera $P(y_i \mid \vec{x}_i; \vec{w})$. Pour ce faire, on change la fonction d'activation et utilisons la fonction **sigmoïde** $\sigma(a) = \frac{1}{1+e^{-a}}$. On peut dériver cette fonction à partir du théorème de Bayes précédemment :

$$
\begin{aligned}
p(y_i = 1 \mid \vec{x}_i) &= \frac{p(\vec{x}_i \mid y_i = 1)p(y_i = 1)}{p(\vec{x}_i \mid y_i = 1)p(y_i = 1) + p(\vec{x}_i \mid y_i = 0)p(y_i = 0)} \\
&= \frac{1}{1 + \frac{p(\vec{x}_i|y_i=0)p(y_i=0)}{p(\vec{x}_i|y_i=1)p(y_i=1)}} \\
&= \frac{1}{1 + \exp\left[\log \frac{p(\vec{x}_i|y_i=0)p(y_i=0)}{p(\vec{x}_i|y_i=1)p(y_i=1)}\right]} \\
\sigma(a) &= \frac{1}{1 + e^{-a}} \\
\nabla_a \sigma &= \sigma(a)(1 - \sigma(a)).
\end{aligned}
$$

On remplace aussi la fonction de perte $\mathcal{L}_{percep}$ par une fonction basée sur la distribution de Bernouilli appelée la *log loss* ou l'entropie croisée :

$$
\begin{aligned}
\mathcal{L}(\mathcal{D}; \vec{w}) &= \sum_i^n y_i \log(\sigma(\langle\langle\rangle\rangle)) + (1 - y_i) \log(1 - \sigma(\langle\langle\rangle\rangle)) \\
\nabla_{\vec{w}} \mathcal{L} &= \sum_i^n (y_i - \sigma(\langle \vec{w}, \vec{x}_i \rangle))\vec{x}_i.
\end{aligned}
$$

Softmax

On peut généraliser la régression logistique à deux classes au cas multiclasses à l'aide de la fonction *softmax*. Tout d'abord, on modifie le réseau pour que la dernière couche contienne autant de neurones que de classes (c.f. Fig. 1.5b). En pratique, on remplace le vecteur de poids par une matrice $\mathbf{W} = \{\vec{w}_1, \ldots, \vec{w}_K\}$ où K est le nombre de classes et utiliserons un produit matriciel $\mathbf{W}\vec{x}$ au lieu d'un produit scalaire. Finalement, la fonction *sigmoïde* est remplacée par la fonction *softmax* :

$$
\begin{aligned}
f_{\vec{w}}(\vec{x}_i) &= softmax(\mathbf{W}\vec{x}_i) = \{f^1_{\vec{w}}(\vec{x}_i), \ldots, f^K_{\vec{w}}(\vec{x}_i)\} \\
f^k_{\vec{w}}(\vec{x}_i) &= \frac{e^{\vec{w}_k \vec{x}_i}}{\sum_{j=1}^K e^{\vec{w}_j \vec{x}_i}}.
\end{aligned}
$$

En sortie, on obtient un vecteur à K dimensions où chaque élément représente la probabilité que $\vec{x}_i$ d'appartenir à la classe $y_c = P(y_c \mid \vec{x}_i)$. Au moment de la prédiction, on prendra $y_c = \arg\max_c f^c_{\vec{w}}(\vec{x}_i)$ pour trouver la classe la plus probable.

On définit le gradient pour chaque sortie du réseau comme suit :

$$\frac{\partial f^k_{\vec{w}}(\vec{x}_i)}{\partial \vec{w}_l} = \frac{\partial(\frac{e^{\vec{w}_k \vec{x}_i}}{\sum_j e^{\vec{w}_j \vec{x}_i}})}{\partial \vec{w}_l}). \tag{1.5}$$

Dans le cas où $k = l$ on a :

$$\begin{aligned}
\frac{\partial f^k_{\vec{w}}}{\partial \vec{w}_l} &= \frac{\partial \frac{e^{\vec{w}_k \vec{x}_i}}{\sum_j e^{\vec{w}_j \vec{x}_i}}}{\partial \vec{w}_l} \\
&= \vec{x}_i \frac{e^{\vec{w}_k \vec{x}_i} \sum_j e^{\vec{w}_j \vec{x}_i} - e^{\vec{w}_l \vec{x}_i} e^{\vec{w}_k \vec{x}_i}}{(\sum_j e^{\vec{w}_j \vec{x}_i})^2} \\
&= \vec{x}_i \frac{e^{\vec{w}_k \vec{x}_i}}{\sum_j e^{\vec{w}_j \vec{x}_i}} \frac{\sum_j e^{\vec{w}_j \vec{x}_i} - e^{\vec{w}_l \vec{x}_i}}{\sum_j e^{\vec{w}_j \vec{x}_i}} \\
&= \vec{x}_i f^k_{\vec{w}}(\vec{x}_i)(1 - f^l_{\vec{w}}(\vec{x}_i)).
\end{aligned}$$

Dans le cas $k \neq l$ on a :

$$\frac{\partial f^k_{\vec{w}}}{\partial \vec{w}_l} = -\vec{x}_i \frac{e^{\vec{w}_l \vec{x}_i}}{\sum_j e^{\vec{w}_j \vec{x}_i}} \frac{e^{\vec{w}_k \vec{x}_i}}{\sum_j e^{\vec{w}_j \vec{x}_i}}.$$

Finalement, on utilisera l'entropie croisée vue plus haut comme fonction de perte $\mathcal{L}$:

$$\mathcal{L}(\mathcal{D}, \mathbf{W}) = -\sum_i^n \sum_k^K \vec{y}_i \log(f^k_{\vec{w}}(\vec{x}_i)).$$

On définit son gradient comme suit :

$$\begin{aligned}
\nabla_{\vec{w}_l}\mathcal{L} &= -\sum_{i}^{n}\sum_{k}^{K}\vec{y}_i\frac{\partial \log(f^k_{\vec{w}_l}(\vec{x}_i))}{\partial a_l} \\
&= -\sum_{i}\vec{x}_i(\sum_{k}^{K}\vec{y}_{ik}(1-f^l_{\vec{w}}(\vec{x}_i)) - \sum_{k\neq l}\vec{y}_{ik}f^l_{\vec{w}}(\vec{x}_i)) \\
&= \sum_{i}\vec{x}_i(\sum_{k}^{K}-\vec{y}_{ik} + -\vec{y}_{ik}f^l_{\vec{w}}(\vec{x}_i)) - \sum_{k\neq l}\vec{y}_{ik}f^l_{\vec{w}}(\vec{x}_i)) \\
&= \sum_{i}\vec{x}_i(f^l_{\vec{w}}(\vec{x}_i) - \vec{y}_{il}),
\end{aligned}$$

où $a_k = \vec{w}_k\vec{x}_i$.

1.4.2 Perceptron multicouche

Il est possible de combiner plusieurs couches au sein d'un perceptron pour approximer des fonctions non linéaires et le rendre plus puissant. Pour le cas multiclasses avec un réseau à L couches, on a donc :

$$f_{\mathbf{W}}(\vec{x}_i) = \text{softmax}(\mathbf{W}^L h^{L-1}(\dots h^1(\mathbf{W}^1\vec{x}_i))\dots)$$

où $\mathbf{W}^l$ représente une matrice de poids pour la couche l et h^l est la fonction d'activation à la couche l. Dans le cas deux classes, on remplacera $softmax$ par la fonction sigmoïde et la matrice $\mathbf{W}^L$ sera remplacée par un vecteur.

1.4.3 Entraînement par descente de gradient

Comme vu auparavant, on entraîne un réseau de neurones par descente de gradient. On calculera le gradient de la fonction de perte par rapport à chaque matrice $\mathbf{W}^L$ du réseau. Tout d'abord, le modèle $f_{\mathbf{W}}(\vec{x}_i)$ fait la prédiction pour une entrée $\vec{x}_i$, opération appelée la propagation avant. Ensuite, l'algorithme de rétropropagation permet de calculer les gradients par rapport à la fonction de perte $\mathcal{L}$.

Rétropropagation

L'algorithme de rétropropagation n'effectue pas la mise à jour des poids, il ne fait que calculer les gradients $\nabla_{\mathbf{W}}\mathcal{L}$ pour chaque couche du réseau. Pour ce faire, on utilise les activations $a^k = \mathbf{W}^k J^{k-1}$ pour chaque couche k où $J^k = h(a^k)$ est l'activation de la couche k suivie de la fonction de non-linéarité h. Comme on le voit à l'algorithme 2, on calcule le gradient de la couche k en utilisant le gradient de la couche précédente.

Input: Un couple $(\vec{x}, y)$
Input: Une prédiction $\hat{y}$
$g \leftarrow \nabla_{\hat{y}}\mathcal{L}(\hat{y}, y)$;
for $k = l, l-1, \ldots, 1$ **do**
  On multiplie point à point le gradient g avec le gradient de la fonction d'activation.;
  $g \leftarrow g \odot h'(a^k)$;
  $\nabla_{\mathbf{W}^k}\mathcal{L} = gJ^{(k-1)T}$;
  $g \leftarrow \mathbf{W}^{(k)T} g = \nabla_{J^{k-1}}\mathcal{L}$;
end

Algorithm 2: Algorithme de rétropropagation pour un réseau de neurones multicouches[33].

Une fois tous les gradients calculés, on obtient les gradients $\nabla_{\mathbf{W}^k}\mathcal{L}$ pour chaque couche du réseau. On peut alors mettre à jour les poids pour l'itération $t+1$ grâce à l'équation 1.6 qui est semblable à l'équation 1.3 :

$$\mathbf{W}^{l(t+1)} = \mathbf{W}^{l(t)} - \alpha(\nabla_{\mathbf{W}^{l(t)}}\mathcal{L}). \tag{1.6}$$

1.4.4 Réseaux à convolution

Un problème avec les réseaux multicouches traditionnels est leur grand nombre de paramètres. En effet, si on voulait utiliser un réseau à 2 couches cachées composées respectivement de 256 et 10 neurones pour analyser des images vectorisées de 784 pixels on obtiendrait un réseau avec plus de 200 000 paramètres. Les réseaux à convolutions [56] permettent de mitiguer ce problème et aussi de construire de plus profondes architectures. Ces réseaux sont spécialisés pour les données structurées en grilles telles que les images ou les séries temporelles. Comme leur nom l'indique, les

réseaux à convolution utilisent une opération bien connue en traitement de signal, la **convolution**.

Convolution

L'opération de convolution (*) permet d'appliquer un filtre à un signal. Soit une entrée (signal) I et un noyau (filtre) K. En apprentissage profond, le résultat de la convolution $I*K$ est appelé une carte d'activation. L'entrée est un tenseur de données et le noyau est un tenseur de paramètres. On note que chacune de ces fonctions est nulle partout en dehors du tenseur défini. Voici un exemple du résultat d'une convolution sur une entrée 2D et un noyau de taille $M \times N$:

$$S(i,j) = (K * I)(i,j) = \sum_{m}^{M} \sum_{n}^{N} I(i-m, j-n)K(m,n). \tag{1.7}$$

Il est à noter que contrairement à la théorie où l'on doit transposer le noyau avant de l'appliquer, la plupart des bibliothèques d'apprentissage profond ne font pas cette opération. On appelle alors cette opération la **cross-corrélation**. Malgré cette différence, par abus de langage on appelle tout de même cette opération une convolution dans ce document. Une propriété intéressante de transposer le noyau est que la convolution est commutative :

$$S(i,j) = (I * K)(i,j) = \sum_{m}^{M} \sum_{n}^{N} I(m,n)K(i-m, j-n).$$

En pratique, l'équation 1.7 est utilisée, car elle est plus rapide.

La principale motivation des couches à convolution est de réduire le nombre de paramètres dans les réseaux. Par exemple, on voit qu'à Fig. 1.6 le nombre de paramètres requis est moindre pour la convolution que pour la couche pleinement connectée. De plus, les réseaux à convolution sont invariants à la translation. En effet, si l'on translate l'entrée I, la sortie S sera translatée de la même façon.

Bien qu'en théorie la convolution est une intégrale sur deux fonctions, elle n'est pas définie lorsqu'une des deux fonctions n'est pas définie. Les réseaux à convolution utilisant des entrées et des noyaux de tailles finies, on ne peut pas calculer le résultat de la convolution aux abords de l'entrée. Ainsi, la taille de la sortie est réduite de

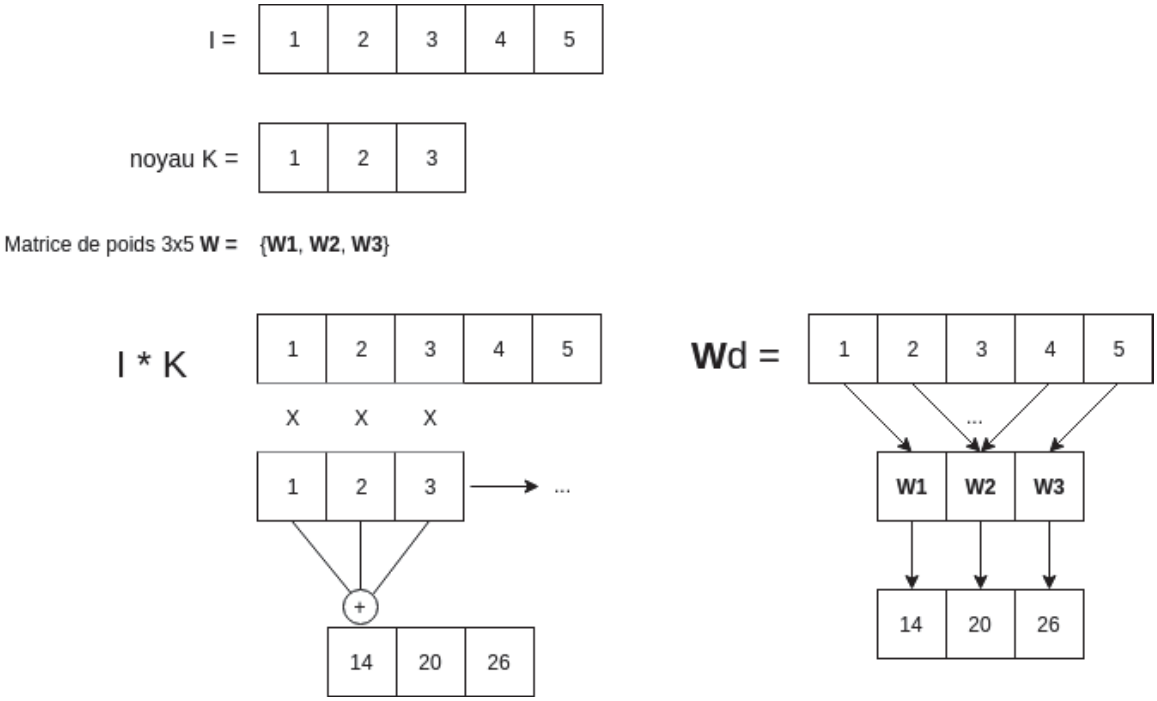


figure 1.6 – [Gauche] Exemple d'application de la convolution «valide» sur l'entrée I et le noyau K. [Droite] Exemple d'application d'une couche pleinement connectée. On remarque que cette méthode utilise 5 fois plus de paramètres que la convolution.

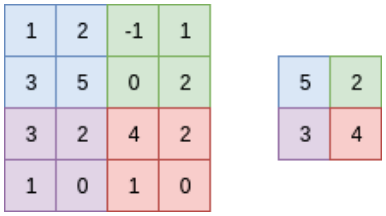


figure 1.7 – Résultat d'un sous-échantillonnage maximum. Dans cet exemple, la taille du noyau est de 2×2 et le pas de glissement est de 2.

$M/2, N/2$ par rapport à la taille de l'entrée. Pour garder la même taille, on tamponne l'entrée avec des zéros avant d'appliquer la convolution. C'est ce qu'on appelle une convolution «similaire» contrairement à une convolution dite «valide».

On aimerait garder la même taille pour plusieurs raisons. Tout d'abord, on peut créer des réseaux plus profonds, ce qui améliore les performances. Deuxièmement, les réseaux à convolution modernes fusionnent plusieurs cartes d'activation obtenues à partir de noyaux différents. Par exemple dans le cas de *Inception*[100], on utilise plusieurs noyaux de tailles $1 \times 1, 3 \times 3$ et 5×5 pour obtenir une carte d'activation finale. Sans la convolution «similaire», il serait difficile de créer de telles architectures.

1.4.5 Sous-échantillonnage

Le sous-échantillonnage (*Pooling*) permet de rendre la représentation des données invariante aux petites translations et a l'avantage de réduire la résolution des données(c.f., figure 1.7). Ces avantages améliorent grandement l'efficacité du réseau sans en affecter les performances. L'opération s'exécute de la manière suivante : on glisse une fenêtre de dimension variable sur les données d'entrée. Le pas du glissement se fait aussi de manière variable. On utilise en général des fenêtres de taille 2×2 et un pas de 2. On applique alors une opération sur cette fenêtre pour obtenir un scalaire. On peut prendre le maximum, la moyenne ou encore la médiane. La méthode d'échantillonnage diffère selon les architectures, mais on utilise généralement le maximum.

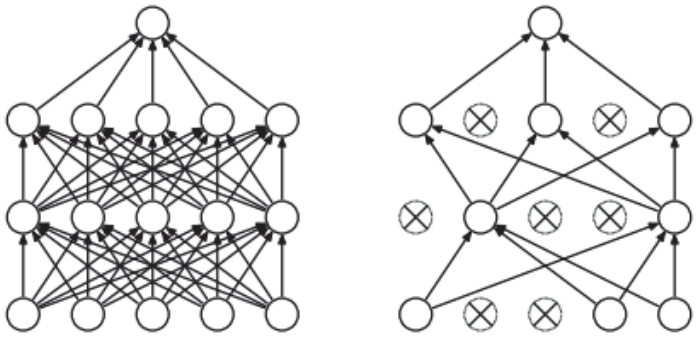


figure 1.8 – Exemple de mise à zéro des connexions. Image reprise de [98] (*Creative Commons*).

1.4.6 Dropout

La méthode de régularisation Dropout[98] est une méthode peu coûteuse pour réduire le sur apprentissage. Cette méthode force à zéro certains poids choisis au hasard durant l'entraînement (c.f., figure 1.8). Ainsi, on empêche la surspécialisation des neurones, car les entrées qu'ils reçoivent sont bruitées.

On peut voir cette technique comme l'entraînement d'un ensemble de modèles, car plusieurs configurations de réseaux sont utilisées lors de l'entraînement. Durant la phase de test, toutes les connexions sont activées.

1.4.7 Normalisation par lot

La normalisation par lot (BatchNorm)[45] a pour objectif de normaliser les activations entre les couches. L'objectif de cette opération est de faciliter l'apprentissage de réseaux plus profonds. En effet, les couches sont mises à jour simultanément alors que le gradient indique comment modifier chaque paramètre en présumant que les autres restent constants. Cela peut mener à des mises à jour qui n'améliorent pas le modèle. La normalisation par lot permet de mitiguer ce problème en normalisant les entrées. Soit une carte d'activations a, on obtient sa version normalisée a' en centrant

et réduisant la carte :

$$\hat{a} = \frac{a - \mu}{\sigma}$$
$$a' = \gamma \hat{a} + \beta$$

où γ et β sont des paramètres appris qui permettent aux résultats d'avoir une moyenne et variance variable. Le but de ces paramètres est de pouvoir représenter les mêmes fonctions qu'auparavant, mais en ayant une dynamique d'apprentissage plus facile et plus stable. La moyenne μ et la variance σ^2 sont calculées sur le lot en cours. Lors de l'inférence, on utilisera une moyenne mobile apprise sur l'ensemble d'entraînement pour approximer ces valeurs. On suggère d'utiliser plusieurs exemples par lot lors de l'entraînement pour avoir une meilleure estimation de celles-ci.

1.4.8 Saut de connexion (*skip connections*)

Les sauts de connexion permettent à l'information et au gradient de mieux se propager dans le réseau. Cette technique consiste à prendre des cartes d'activations au début du réseau et de les ajouter aux cartes d'activations situées plus profondément dans le réseau. Dans un contexte d'*encodeur-décodeur* (dont nous verrons les détails à la Section 1.5) comme on le voit à la figure 1.13, ceci permet au décodeur d'obtenir de l'information plus riche et ainsi faire de meilleures prédictions.

1.4.9 Connexion résiduelle (*residual connections*)

Augmenter la profondeur d'un réseau est un bon moyen pour augmenter ses performances. Malheureusement, les réseaux très profonds sont extrêmement difficiles à entraîner. En effet, leurs performances stagnent puis se dégradent rapidement durant l'entraînement. Pour régler ce problème, les connexions résiduelles[40], telles que présentées à la figure 1.9, ajoutent des connexions avant et après les blocs ce qui permet à l'information de se propager sans transformation.

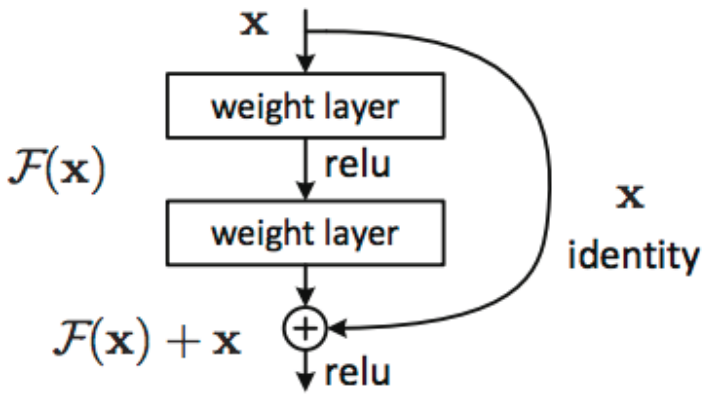


figure 1.9 – Bloc avec une connexion résiduelle. Image prise de [40] aussi disponible sur arXiv en *Creative Commons*.

1.4.10 Convolution dilatée

On définit le champ réceptif comme la portion de l'entrée à laquelle un neurone a accès. Celui-ci permet au réseau de neurones de réagir à des caractéristiques globales ou locales. Bien que celui-ci augmente linéairement avec la profondeur de l'image, il reste tout de même minime lorsqu'on utilise des images de grandes tailles. Pour augmenter le champ réceptif, on peut augmenter la taille des noyaux utilisés ou encore augmenter la profondeur du réseau. Malheureusement, ces solutions affectent la vitesse d'exécution ainsi que le nombre de paramètres à apprendre. Les convolutions dilatées [111] offrent une alternative intéressante à ce problème.

Celles-ci *dilatent* le noyau en augmentant son champ réceptif sans augmenter le nombre de paramètres. Comme on le voit à la figure 1.10, on peut couvrir une plus grande partie de l'image avec le même nombre de paramètres.

1.4.11 Construction de réseaux à convolution

Les réseaux à convolution sont des réseaux de neurones qui utilisent des couches de convolution. L'architecture typique en classification combine des couches de convolutions et d'échantillonnage puis vectorise le résultat pour ensuite utiliser des couches pleinement connectées et faire la prédiction. Par exemple, à la Fig. 1.11 on a l'archi-

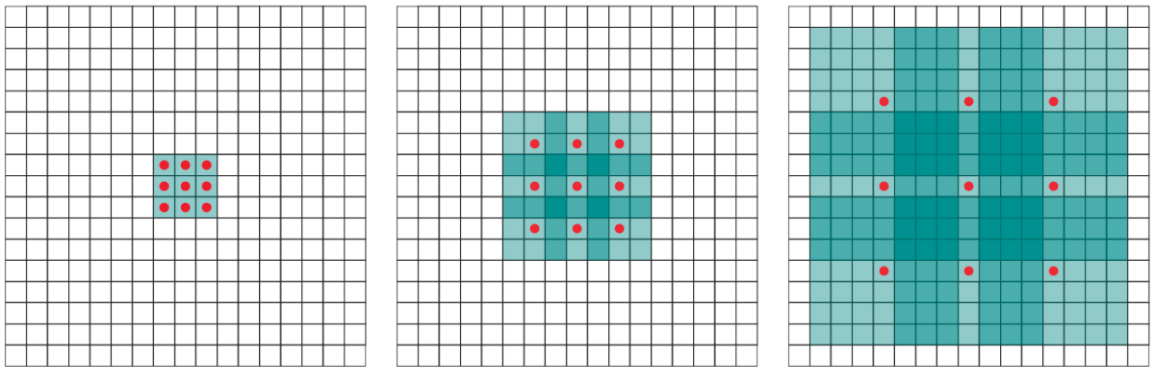


figure 1.10 – Représentation du champ réceptif lorsqu'on utilise des convolutions dilatées. a) Noyau sans dilatation. b) Noyau avec une dilatation de 1. c) Noyaux avec une dilatation de 4. Image prise de [111] aussi disponible sur arXiv en *Creative Commons*.

tecture d'un des premiers réseaux à convolution le VGG-16[95]. Il contient 16 couches dont 13 à convolution. On remarque la réduction de taille introduite par les modules de sous-échantillonnages. Cette réduction permet d'utiliser des couches pleinement connectées (*fully-connected* ou *fc*) pour effectuer la classification.

Cette façon de faire a le défaut d'être dépendante de la taille de l'image en conséquence de la vectorisation et des couches pleinement connectées. La convolution pouvant être appliquée sur une taille d'entrée variable, on peut construire des réseaux faits entièrement de convolutions (*fully convolutional* en anglais). Ces réseaux peuvent ainsi travailler sur des entrées de tailles variables. Pour ce faire, au lieu de vectoriser la sortie des convolutions, on ajoute une dernière convolution avec le nombre de classes comme nombre de filtres. Finalement, un sous-échantillonnage est appliqué sur l'entièreté de la carte d'activation ce qui donnera la prédiction pour chaque classe.

1.5 Segmentation

L'objectif de la segmentation est de classifier chaque pixel d'une image. On utilise généralement la cross-entropie moyenne sur l'ensemble des pixels comme objectif, mais on peut ajouter des objectifs sur les contours[68] ou alors sur l'uniformité des

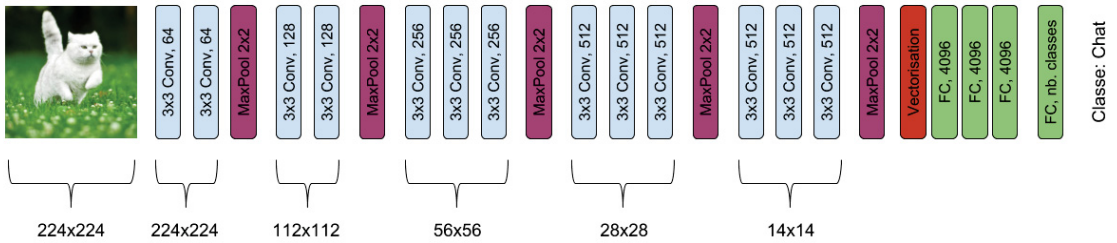


figure 1.11 – Architecture du VGG-16[95]. Les couches de convolution sont décrites par la taille de leur noyau et du nombre de filtres. Les couches *fc* sont des couches pleinement connectées. Celles-ci sont décrites par leur nombre de neurones.

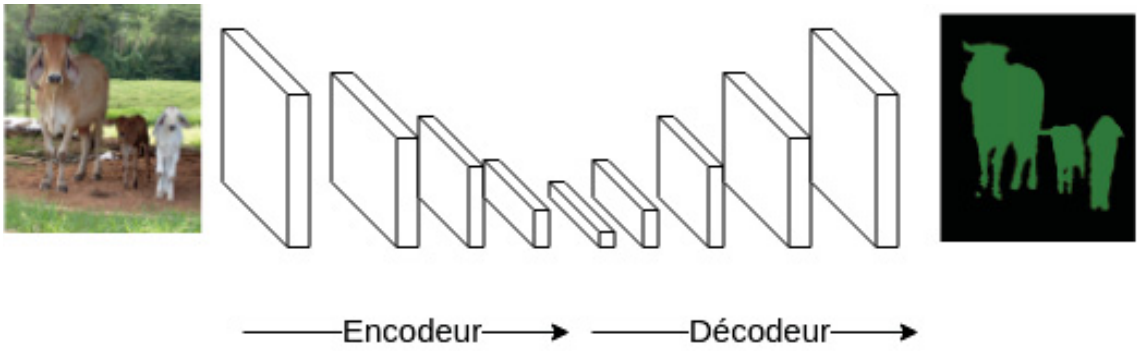


figure 1.12 – Architecture du ConvDeconv.

résultats[37].

1.5.1 ConvDeconv

Le réseau ConvDeconv[78] est l'un des premiers réseaux de segmentation à utiliser une architecture *encodeur-décodeur*. Il incorpore des convolutions transposées permettant d'agrandir la résolution de l'image en utilisant des convolutions. Ce modèle adopte une architecture dite encodeur-décodeur où l'on réduit la dimension spatiale puis on retrouve la dimension originale en utilisant des convolutions transposées ou des méthodes d'agrandissement standards(c.f. Fig. 1.12). Cette méthode a pour défaut «d'oublier» des détails lors de la réduction spatiale. Lors de sa publication, ce modèle était état-de-l'art sur Pascal VOC 2007[27].

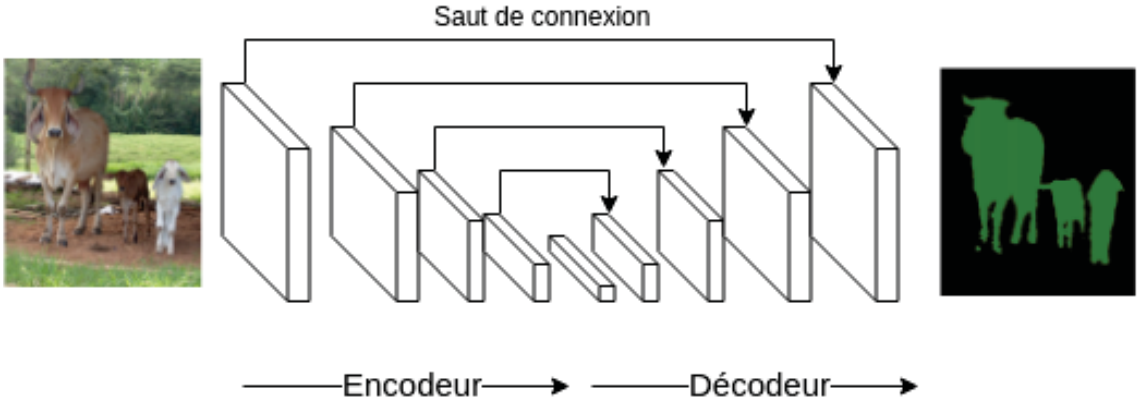


figure 1.13 – Architecture du U-Net.

1.5.2 U-Net

Le U-net[89] est le successeur du ConvDeconv. Il ajoute des sauts de connexions pour connecter les couches de même dimension (c.f. Fig. 1.13). Cela a pour effet d'augmenter les détails présents, car l'information est mieux propagée dans le réseau. Ce réseau a tout d'abord été utilisé pour la segmentation médicale, mais a rapidement été utilisé dans d'autres domaines[12, 71, 4].

1.6 Localisation

La localisation a pour objectif de résoudre deux tâches, la détection d'objets, c'est-à-dire identifier les objets dans l'image en prédisant une boîte autour de chaque objet puis classifier l'objet en lui-même. Les algorithmes d'apprentissage profond ont profondément changé ce domaine en obtenant de bien meilleures performances. En particulier, les réseaux à convolution ont permis de créer des applications de localisation en temps réel et très performantes. Les trois principaux modèles sont Faster R-CNN, YOLO et SSD.

1.6.1 Faster R-CNN

Faster R-CNN[87] est le successeur du Fast R-CNN[31], dont la principale contribution est l'ajout d'*ancres*. Les modèles antérieurs effectuent une régression complète

sur les dimensions des boîtes. Ren et *et al.*[87] proposent de décomposer le problème en deux sous-problèmes : on cherche d'abord grossièrement la position des objets dans l'image pour ensuite la raffiner. La recherche grossière se fait à partir des boîtes les plus communes de l'ensemble de données. Ces boîtes sont nommées *ancres*.

En pratique, le modèle est décomposé en trois étapes. Premièrement, un *Region Proposal Network (RPN)* extrait des propositions de boîtes (grâce aux ancres). On obtient ainsi plusieurs régions contenant les cartes d'activation à l'intérieur des ancres actives. Pour unifier ces régions, on utilise l'algorithme de *RoI Pooling* qui sépare chaque région en un nombre fixe de sous-régions. Puis on effectue un sous-échantillonnage maximal pour unifier les régions. Finalement, on prédit la classe et les points extrêmes de chacune des régions (c.f. Fig. 1.14a).

Ce modèle a révolutionné le domaine de la localisation, car il était le premier à proposer des ancres pour ses propositions. Cette innovation facilite l'entraînement tout en améliorant les résultats.

1.6.2 You Only Look Once (YOLO)

Le modèle *You only look once (YOLO)*[84] est un modèle intéressant, car il est le premier a n'utiliser qu'un seul réseau pour faire sa prédiction. Cette innovation lui permet d'être le premier réseau de localisation à pouvoir être utilisé en temps réel. Pour ce faire, le réseau prédit une grille de taille $N \times N$ où chaque cellule de la grille peut prédire jusqu'à B boîtes. Chaque boîte est composée des coordonnées de la boîte (position, hauteur et largeur), la probabilité qu'une boîte se trouve à cet endroit ainsi que de la classe associée. Un algorithme de suppression des non-maximums est ensuite appliqué aux prédictions(c.f. Fig. 1.14b).

Son successeur YOLOV2[85] ajoute des ancres semblables à Faster R-CNN. Le modèle prédit la boîte par défaut ainsi qu'un décalage. Cette technique améliore les performances ainsi que l'entraînement tout en gardant sa rapidité d'exécution.

1.6.3 Single-Shot Detector (SSD)

Le *Single-Shot Detector (SSD)*[63] est similaire à YOLOV2. Il est composé d'un seul réseau qui effectue la prédiction pour toute l'image en utilisant des ancres. Sa

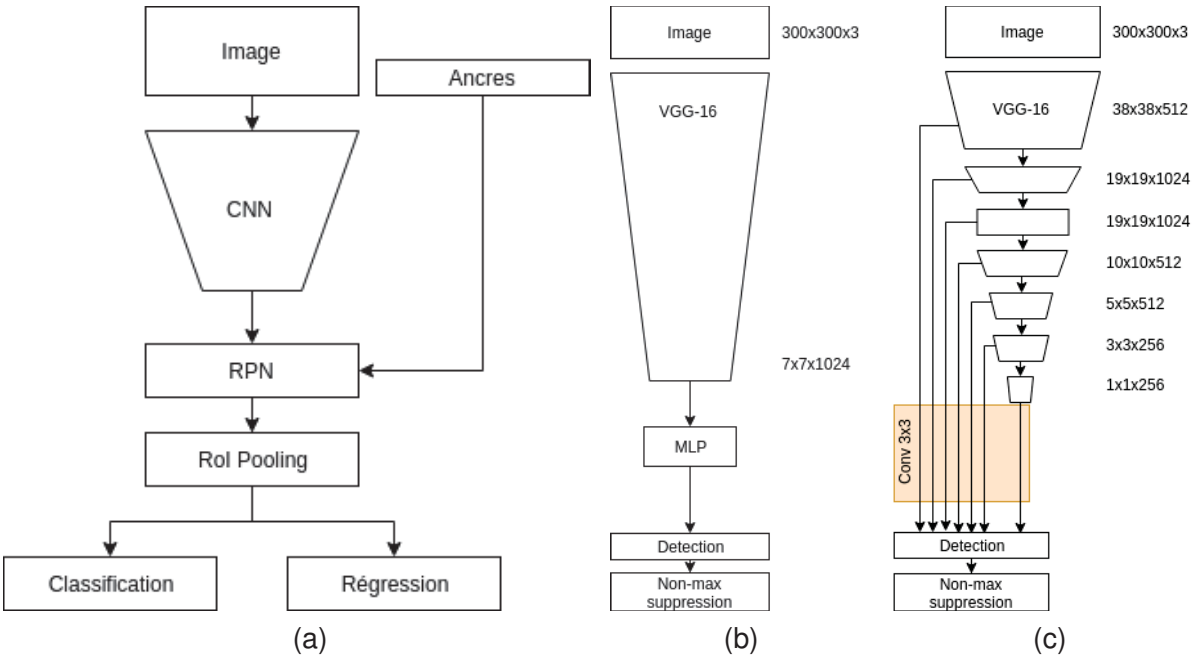


figure 1.14 – a) Architecture du Faster R-CNN. RPN représente le *Region Proposal Network*. b) Architecture de YOLO. *MLP* représente un réseau de neurones multi-couches. c) Architecture du Single-Shot Detector (SSD).

principale contribution est d'utiliser des cartes d'activations à plusieurs niveaux (c.f., Fig. 1.14c). De plus, il n'utilise pas de couches de neurones entièrement connectées ce qui réduit son nombre de paramètres. En conséquence, il plus performant que YOLOV2 tout en pouvant traiter 60 images de tailles 300×300 par secondes.

Chapitre 2

Estimation de la complexité des bases de données

L'estimation de complexité d'un jeu de données est un premier pas vers la compréhension de celui-ci. En effet, elle permet de mieux comprendre les relations inter et intra classes, la densité des données et la complexité de la frontière entre les classes. Ces caractéristiques affectent la difficulté à apprendre les frontières séparant les classes.

Lorena et *et al.*[64] propose de catégoriser les méthodes existantes en six catégories que nous verrons dans ce chapitre : chevauchement de caractéristiques, linéarité, analyse du voisinage, analyse par graphe, dimensionnalité des données et balancement des classes. On appelle ces méthodes les "*c-measures*" pour *complexity measures*. À la fin du chapitre, nous verrons quelques applications possibles de ces méthodes.

2.1 Notation

Soit un ensemble de données $\mathcal{D} = \{(\vec{x}_0, y_0), \ldots, (\vec{x}_n, y_n)\}$ avec y_i une étiquette de classe parmi $\{c_1, \ldots, c_{n_c}\}$ et $\vec{x}_i \in \mathbb{R}^d$ un élément composé de d caractéristiques.

$kNN(\vec{x}_i)$ rapporte la classe de l'exemple $\vec{x}_i$ produite par l'algorithme du «k plus proche voisins»(*k-nearest neighbors*) sur l'ensemble $\mathcal{D}$. Par exemple $1NN(\vec{x}_i)$ est le résultat du kNN pour $k = 1$.

$\mathbb{1}(a)$ est la fonction indicateur qui vaut 1 lorsque x est vrai et 0 sinon :

$$\mathbb{1}(a) = \begin{cases} 1, \text{ si } a \text{ est vrai} \\ 0, \text{ sinon} \end{cases} .$$

2.2 Chevauchement des caractéristiques

Ces *c-measures* analysent l'importance des caractéristiques. Plus les caractéristiques sont discriminantes, plus il est facile de résoudre le problème. À noter que les métriques de chevauchement ne sont pas idéales pour les images. En effet, les images sont caractérisées par des combinaisons hautement non linéaires de leurs pixels.

2.2.1 F1

Premièrement, $F1$ calcule le ratio maximal du discriminant de Fischer :

$$F1 = \max_{i=1}^{d} \frac{\sum_{j=1}^{n_c} n_{c_j} (\mu_{c_j}^{d_i} - \mu^{d_i})^2}{\sum_{j=1}^{n_c} \sum_{l=1}^{n_{c_j}} (x_{li}^{j} - \mu_{c_j}^{d_i})^2}$$

,

où $\mu_{c_j}^{d_i}$ est la moyenne de la caractéristique d_i pour la classe c_j, μ^{d_i} est la moyenne de la caractéristique d_i sur l'ensemble et x_{li}^{j} est la caractéristique i d'un exemple de la classe c_j. Cette mesure permet d'estimer la dispersion des caractéristiques dans l'espace. Bien entendu, cette méthode fonctionne mieux si les distributions sont normales, ce qui n'est pas souvent le cas avec des images.

2.2.2 F2

$F2$ calcule l'*intersection over union (IoU)* (voir Chapitre 1) pour chaque paire de classes et pour chaque caractéristique. Dans le cas 2 classes, on calcule le score $F2$ par le produit des *IoUs* :

$$F2 = \prod_{i}^{d} \frac{intersection(d_i)}{union(d_i)}.$$

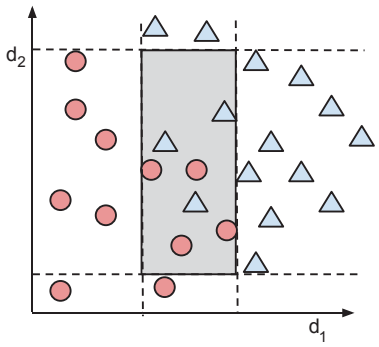


figure 2.1 – Calcul de l'intersection entre deux classes pour deux caractéristiques d_1 et d_2.

Pour le cas multiclasses, on calculera la F2 entre chaque paire de classe et gardera la valeur maximale. À la Fig. 2.1, on peut voir comment calculer l'IoU.

2.2.3 F3

$F3$ permet de trouver la caractéristique la plus efficace pour le cas deux classes. L'intuition de cette métrique est qu'une caractéristique avec une petite zone chevauchante est discriminante et peut donc aider l'apprentissage. On calcule F3 comme suit :

$$F3 = \max_{1 \leq i \leq d} \frac{n - intersection(d_i)}{n},$$

où n est le nombre total d'exemples et $intersection(d_i)$ calcule le nombre d'exemples qui sont dans la région chevauchante de la caractéristique d_i. Lorsque $F3 = 1$, cela signifie qu'une caractéristique ne possède pas de zone chevauchante. Il est donc extrêmement facile de séparer les deux classes. À l'opposé, $F = 0$ indique que l'ensemble est totalement mélangé. Pour que F3 soit efficace, il faut que la frontière de décision soit perpendiculaire aux axes, sinon elle est inutile.

2.2.4 F4

$F4$[81] est la version itérative de $F3$. En appliquant $F3$, on obtient la caractéristique la plus efficace, on enlève alors tous les exemples qui peuvent être classifiés par

celle-ci pour obtenir l'ensemble $\mathcal{D}_1$. Itérativement, on applique l'algorithme l fois sur le nouvel ensemble et on obtient un nouvel ensemble de données $\mathcal{D}_l$. $F4$ est obtenu en appliquant $F3$ sur ce dernier. Par défaut, $l = 1$.

2.3 Linéarité

Ces *c-measures* estiment la linéarité de la frontière de décision, plus la frontière est linéaire, plus elle est facile à apprendre pour un algorithme d'apprentissage. Pour ce faire, on aura besoin d'un classificateur linéaire. Ho et Basu[41] proposent de résoudre un problème d'optimisation alors que Orriols et *et al.*[81] utilisent un SVM.

2.3.1 L1

La somme des erreurs ou $L1$ est la moyenne des distances entre la frontière de décision apprise par l'algorithme et les données mal-classées :

$$L1 = \frac{1}{n}\sum_{i=1}^{n}\epsilon_i,$$

où ϵ_i est la distance entre l'exemple mal-classé x_i et la frontière de décision apprise.

2.3.2 L2

L2 est le taux d'erreur du classificateur linéaire. On entraînera un SVM sur l'ensemble avant de calculer le taux d'erreur suite à l'entraînement.

2.3.3 L3

$L3$ est similaire à $L2$, mais on génère l'ensemble de test là où $L2$ utilise un ensemble de test prédéfini. Comme on le voit à la Fig. 2.2, on interpole deux points d'une même classe pour créer un point de l'ensemble de test. Suite à la création d'un certain nombre de points par classe, on calcule l'erreur de classification sur cet ensemble.

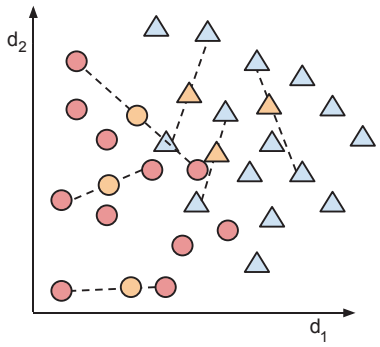


figure 2.2 – Processus de créations des nouveaux éléments utilisés dans L3.

2.4 Analyse du voisinage

Ces mesures caractérisent la difficulté de la tâche en fonction du voisinage de chaque élément $\vec{x}_i$. Ces méthodes sont souvent sensibles au bruit et demandent la construction de structure telle que des arbres. Une importante source d'erreurs pour les images est la nécessité de trouver une bonne méthode pour déterminer les voisins. La distance euclidienne a souvent peu de sens sur une image faite de pixels.

2.4.1 N1

N1 calcule le ratio entre le nombre d'exemples dont au moins un voisin est d'une classe différente et le nombre total d'exemples. Pour trouver les voisins, on calcule d'abord un *minimal spanning tree (MST)*, les noeuds connectés sont alors voisins :

$$N1 = \frac{1}{n}\sum_{i}^{n}\sum_{j\neq i}\mathbb{1}(e_{ij} \in MST \wedge y_i \neq y_j)$$

où e_{ij} est le lien entre $\vec{x}_i$ et $\vec{x}_j$.

2.4.2 N2

N2 mesure le ratio entre les voisins de classes différentes et ceux de même classe. Pour ce faire, nous comparons la distance entre $\vec{x}_i$ et son voisin de même classe le plus près à la distance entre $\vec{x}_i$ et son voisin de classe différente le plus près. Si ce dernier

est plus près que le voisin de même classe, on suppose que l'espace est hétérogène et donc que le problème est complexe.

$$N2 = \frac{\sum_{i=1}^{n} \delta(\vec{x}_i, 1NN(\vec{x}_i) \in y_i)}{\sum_{i=1}^{n} \delta(\vec{x}_i, 1NN(\vec{x}_i) \in y_j \neq y_i)}$$

où δ est une fonction de distance, souvent la distance euclidienne ou un noyau gaussien.

2.4.3 N3

N3 crée un ensemble de test à partir de l'ensemble d'entraînement. On évalue ensuite le kNN (avec $k = 1$) sur celui-ci :

$$N3 = \frac{\sum_{i=1}^{n_t} I(1NN(x_i) \neq y_i)}{n_t}$$

où n_t est la taille de l'ensemble de test.

2.4.4 N4

N4 est très semblable à L3, mais on remplace le SVM par un 1NN. Ce dernier ne nécessite pas d'entraînement, mais il est davantage affecté par du bruit. Le SVM ne peut caractériser les fonctions non linéaires sans utiliser de noyau, mais il est plus résistant au bruit.

2.4.5 T1

Le ratio d'hypersphères ou T1 est une mesure qui construit une hypersphère autour de chaque point. Le rayon de chaque hypersphère est dilaté jusqu'à ce que celle-ci touche un élément d'une classe différente. On ne garde que les hypersphères qui ne sont pas un sous-ensemble d'une autre :

$$T1 = \frac{\#Hypersphères(\mathcal{D})}{n}$$

où $\#Hypersphères(\mathcal{D})$ est le nombre final d'hypersphères construites sur l'ensemble $\mathcal{D}$. T1 tend vers 1 lorsque le voisinage de chaque point n'est pas uniforme et qu'on ne peut contenir plusieurs points par hypersphère. La valeur minimale est lorsque chaque classe est contenue dans son hypersphère donc $T1 = n_c/n$.

2.5 Analyse par graphes

Certaines *c-measures* caractérisent les données à l'aide de graphes de connectivité. [28] On crée d'abord celui-ci à l'aide du $\epsilon - NN$, où l'on garde un lien si la distance entre les éléments $\delta(\vec{x}_i, \vec{x}_j) \leq \epsilon$. On obtient donc un graphe $G = (V, E)$ où $v_i \in V$ est le noeud représentant $\vec{x}_i$. Un lien $e_{ij} \in E$ représente un lien entre v_i, v_j. Le graphe est ensuite élagué . Pour ce faire, on garde le lien e_{ij} ssi $y_i == y_j$. On note que $|V| = n, 0 \leq |E| \leq \frac{n(n-1)}{2}$. Toutes les mesures ci-dessous requièrent la construction et l'élagage de ce graphe ce qui peut être contraignant pour des bases de données volumineuses.

Ces méthodes reposent sur le concept de densité. Si une base de données possède des régions denses de points appartenant à une même classe, on peut en conclure que le problème est simple. Semblables aux méthodes par voisinages, il faut avoir une méthode appropriée pour sélectionner les voisins lorsqu'on utilise des images.

2.5.1 Densité

On calcule la densité d'un graphe en comparant le nombre de liens retirés lors de l'élagage. Si une majorité de liens sont préservés, cela implique que les classes sont homogènes et donc peu complexes :

$$Density = \frac{2|E|}{n(n-1)}.$$

2.5.2 ClsCoef

ClsCoef regarde si les voisins d'une donnée $\vec{x}_i$ sont connectés entre eux. Si c'est le cas, c'est que la donnée $\vec{x}_i$ se trouve dans une région dense. Si on a en moyenne

des données dans des régions denses, on peut en conclure que la base de données est simple :

$$ClsCoef = \frac{1}{n}\sum_{i=1}^{n}\frac{2|e_{jk} : v_j, v_k \in N_i|}{k_i(k_i - 1)}$$

où N_i est le voisinage d'une donnée x_i et k_i est la cardinalité du voisinage.

2.6 Dimensionnalité des données

Les *c-measures* suivantes estiment la difficulté de la base de données en regardant sa dimensionnalité. On émet ainsi l'hypothèse qu'un problème ayant un grand nombre de dimensions souffre de la malédiction de la dimensionnalité de par le fait que l'espace soit peu dense. Par conséquent, ce problème sera plus difficile. Ces méthodes n'utilisant pas l'information de classe, elles ne peuvent donc pas caractériser la frontière de décision.

2.6.1 T2

Le nombre moyen d'exemples par dimension ou T2 met de l'avant la dispersion des données. Si l'on a peu de données, mais beaucoup de dimensions, il sera plus difficile d'apprendre à résoudre le problème :

$$T2 = \frac{n}{d}.$$

2.6.2 T3

Le nombre moyen d'exemples par dimension de l'analyse en composante principale (ACP) ou T3 utilise une version réduite du nombre de dimensions. On calcule d'abord l'ACP sur l'ensemble de données pour décrire 95% de leur variance. On nomme la nouvelle dimensionnalité d'. On applique alors $T2$ sur celle-ci. Une valeur élevée de T2 indique une tâche simple, car on peut la représenter avec peu de dimensions :

$$T3 = \frac{n}{d'}.$$

2.6.3 T4

Le ratio entre d' et d indique la proportion de dimensions importantes dans l'ensemble. Si le ratio est grand, on a alors un problème plus difficile, car il est difficile de caractériser simplement le problème :

$$T4 = \frac{d'}{d}.$$

2.7 Balancement des classes

Lorsqu'une base de données souffre d'un débalancement du nombre de données par classe, les algorithmes d'apprentissage peuvent être amenés à privilégier les classes les plus populeuses. Les mesures suivantes servent à caractériser le problème. Similaires aux mesures de dimensionnalité présentées à la section précédente, elles ne prennent pas en compte la frontière de décision.

2.7.1 C1

L'entropie de la proportion des classes (ou C1) décrit le débalancement entre classes en utilisant l'entropie de la distribution. Soit $p_i = n_{c_i}/n$ la proportion de la classe i où n_{c_i} est le nombre d'élément dans la classe c_i, on calcule C1 comme suit :

$$C1 = -\frac{1}{\log(n_c)} \sum_{i=1}^{n_c} p_i \log(p_i).$$

Lorsque $C1 = 1$, on a que les probabilités sont uniformes pour toutes les classes.

2.7.2 C2

Le ratio de débalancement ou C2 calcule le débalancement des classes par proportion. Une valeur élevée indique un fort débalancement et donc un plus grand risque de sur apprentissage :

$$C2 = \frac{n_c - 1}{n_c} \sum_{i=1}^{n_c} \frac{n_{c_i}}{n - n_{c_i}}.$$

2.8 Applications

Mis à part l'évaluation de la difficulté du jeu de données, les mesures de complexités ont été appliquées à plusieurs tâches telles que le prétraitement des données, l'aide à l'apprentissage ainsi que le méta-apprentissage.

2.8.1 Prétraitement

Les mesures de complexités ont été appliquées à la sélection de caractéristiques (FS)[80]. À l'aide des techniques énoncées plus haut, notamment F1 et F3, on peut trouver les caractéristiques les plus discriminantes de la base de données. Ceci permet de réduire la dimension de notre base de données et ainsi faciliter la tâche de l'algorithme d'apprentissage.

Il existe des cas où la vitesse d'entraînement est un facteur important. Par exemple, lorsqu'on utilise des algorithmes de séries temporelles, on aimerait s'entraîner sur les données les plus récentes possible. On ne peut donc pas s'entraîner pendant des jours sur nos données. Une façon d'accélérer le processus est de réduire la taille de la base de données en choisissant les meilleurs exemples à utiliser. Leyva et *et al.*[59] propose d'utiliser les mesures de complexités pour prédire quel algorithme de réduction sera le meilleur en fonction des caractéristiques de l'ensemble.

Le débruitage des données préalablement à l'entraînement est une étape importante du nettoyage de données. Cela permet d'obtenir de meilleures performances à peu de frais. Dans leur article, Saez et *et al.*[92] prédisent le filtre de débruitage à utiliser.

2.8.2 Aide à l'apprentissage

Ces mesures peuvent aussi être appliquées à l'analyse d'algorithme d'apprentissage. Par exemple, Macia et *et al.*[69] évaluent plusieurs algorithmes tels que des perceptrons, des réseaux bayésiens et des *Random Forest* sur plusieurs ensembles de données possédant des caractéristiques différentes. Cela permet de mieux comprendre dans quel environnement les algorithmes performent le mieux.

Quiterio et *et al.*[83] utilisent celles-ci pour estimer la difficulté d'ensembles à deux classes dans le but de choisir le classificateur binaire approprié.

Finalement, Brun et *et al.*[10] choisissent dynamiquement l'algorithme d'apprentissage à utiliser en fonction de la difficulté de la base de données.

2.8.3 Meta-apprentissage

Le meta-apprentissage utilise des connaissances déjà apprises sur d'autres bases de données pour résoudre un problème sur une nouvelle base de données. Par exemple, Van et *et al.*[104] estiment les performances possibles d'algorithmes d'apprentissage sur une base de données. Pour ce faire, il compare ce jeu de données avec d'autres dont on connait déjà les performances. Cette comparaison est faite grâce aux mesures de complexité. On peut ensuite estimer les performances qu'un algorithme d'apprentissage peut obtenir. La méthodologie est similaire pour d'autres applications. Garcia et *et al.*[29] prédisent les performances de plusieurs filtres de débruitages sur une base de données en la comparant avec d'autres déjà connues.

Chapitre 3

Présentation des bases de données

Tel que discuté au Chapitre 1, les algorithmes d'apprentissage sont entraînés sur des bases de données. Celles-ci, particulièrement dans le milieu académique, permettent de définir une procédure d'évaluation standardisée pour comparer les méthodes. La création de ces bases de données est coûteuse et prend beaucoup de temps. Depuis quelques années, le nombre de bases de données a explosé, multipliant ainsi les applications de l'apprentissage machine.

Plus précisément, les bases de données de localisation ont fait avancer la recherche en proposant de nouveaux défis tels qu'une plus grande variété d'objets, de tailles et de type de scènes. La classification reste un domaine phare de la vision par ordinateur. Cette application a vu, elle aussi, son nombre de bases de données exploser en proposant des défis plus difficiles notamment en réduisant la taille de l'ensemble d'entraînement.

Dans ce chapitre, nous présentons les différentes bases de données de localisation, notamment notre base de données sur l'analyse de trafic routier MioTCD. Puis, nous présentons les bases de données de classification classiques que nous avons utilisées pour notre article qui porte sur une nouvelle mesure de complexité (c.f. Chapitre 4).

3.1 Miovision Traffic Camera Dataset

Le *Miovision Traffic Camera Dataset* (MioTCD) [66] est une base de données créée à l'occasion de l'atelier *Traffic Surveillance Workshop and Challenge* (TSWC) 2017. La base de données comprend 137,743 images entièrement annotées par plus de 200 annotateurs pour effectuer de la localisation ainsi que de la classification de véhicules filmés par des caméras de surveillances le long d'axes routiers. Celle-ci est divisée en 10 classes communes pour les deux défis : Articulated Truck, Bicycle, Bus, Car, Motorcycle, Non-Motorized Vehicle, Pedestrian, Pickup Truck, Single-Unit Truck et Work Van. Les exemples sont représentatifs des scènes que l'on peut retrouver aux abords des routes nord-américaines. Les images contiennent plusieurs difficultés telles que de faibles résolutions, plusieurs artefacts de compression ainsi qu'un grand déséquilibre entre les classes. En effet, la classe Car représente plus de 90% des exemples, ce qui peut amener les algorithmes à privilégier cette classe. L'article, qu'on peut retrouver en Annexe A, décrit le processus de création de la base de données ainsi que les scores obtenus par les méthodes de localisation et de classification modernes. Une analyse plus approfondie des erreurs de localisation permet de mieux comprendre les avantages et désavantages de chacune des méthodes.

3.1.1 Classification

La tâche de classification (Fig. 3.1) contient 648,959 miniatures de véhicules. En plus des classes citées plus haut, la classe Background a été ajoutée. L'ensemble est divisé 90-10 pour créer les ensembles d'entraînement et de test. Suite au défi, il a été déterminé que celui-ci pouvait être facilement résolu par des réseaux à convolution. En effet, un réseau Xception [14] atteint un *Kappa Score* de 0.9627, ce qui est excellent. On note que la classe Non-Motorized Vehicle est problématique, car elle n'obtient que 77.6% de précision. On peut expliquer cette faiblesse par la grande variance des véhicules dans cette classe et du faible nombre d'exemples pour l'entraînement.

figure 3.1 – Exemples de miniatures des 12 classes de l'ensemble MioTCD. Image reprise de [66].

3.1.2 Localisation

Le défi de localisation (Fig. 3.2) contient 137,743 images. Les participants disposaient de 100,000 images d'entraînement et étaient évalués sur les 37,743 restantes. Pour évaluer les méthodes, la méthodologie de Pascal VOC [27] a été utilisée en acceptant les prédictions ayant plus de 50% d'*IoU (intersection over union)* avec une boîte cible de même classe. Le score final est défini comme la moyenne des précisions (mAP) à plusieurs niveaux de rappel. La classe Motorized vehicle a été ajoutée pour les objets trop petits ou occlus pour être classifiés. Lorsque la cible est de cette classe, les modèles peuvent prédire l'une des classes suivantes sans pénalité : Articulated Truck, Bus, Car, Pickup Truck, Single-Unit Truck et Work Van. Les techniques modernes réussissent plutôt bien le défi, par exemple SSD-500[63] obtient un score de 78% de mAP. En guise de comparaison, les méthodes état-de-l'art sur COCO[61] n'obtiennent guère plus que 70% de mAP.

3.2 Pascal VOC

Pascal VOC [27] est l'une des bases de données les plus populaires en localisation. Elle a subi plusieurs transformations au fil des ans pour la rendre plus complète. L'ensemble est donc passé de 4,000 images en 2008 à 11,500 en 2012. Il est à noter que l'ensemble de test est disponible pour la version 2007, mais qu'il faut maintenant passer par un serveur d'évaluation. L'ensemble possède 20 classes et les images ont

figure 3.2 – Exemples de scènes présentes dans la base de données MioTCD. Image reprise de [66].

figure 3.3 – Exemples de scènes présentes dans COCO (*Creative Commons)*.

été prises dans des environnements naturels provenant du site *Flickr*.

3.3 COCO

COCO [61] est une base de données créée dont le but est de remplacer Pascal VOC. Elle contient plus de 300,000 images, dont 200,000 annotées dans plus de 80 classes. Bien que la segmentation d'instances soit la tâche principale de cet ensemble, les algorithmes de localisation l'utilisent comme nouveau standard. COCO est beaucoup plus difficile et les méthodes état-de-l'art approchent à peine les 70% de mAP. De plus, le protocole d'évaluation est plus exhaustif. En effet, le seuil d'IoU est variable et on calcule le score pour plusieurs tailles d'objets. Ainsi, on obtient un portrait plus complet des méthodes. Par exemple, certains algorithmes génèrent de meilleures boîtes, mais ne trouvent pas les petits objets.

3.4 Classification

L'analyse de complexité décrite au Chapitre 2 utilise la classification comme tâche sur laquelle faire son analyse. Les ensembles ci-dessous sont ceux que nous utiliserons au Chapitre 4.

3.4.1 MNIST

MNIST[57] est l'un des ensembles les plus populaires en vision par ordinateur et comporte 70,000 images en niveaux de gris. Chaque image possède 28×28 pixels et représente un caractère numérique manuscrit entre 0 et 9. La tâche du classificateur est de catégoriser l'image correctement. Même sans apprentissage profond, il était possible d'atteindre les 2% d'erreur sur cette tâche[55]. Les méthodes modernes sont sous les 1% d'erreur [99], mais on trouve toujours une utilité à cet ensemble. En effet, on utilise souvent celui-ci pour les validations préliminaires de nouvelles méthodes.

3.4.2 CIFAR10

CIFAR10 [53] est une base de données d'images naturelles. Elle contient 10 classes : airplane, automobile,bird,cat,deer,dog,frog,horse,ship et truck. La base de données comporte 60,000 images couleur de 32×32 pixels. Depuis 2015, les réseaux à convolution de l'état-de-l'art obtiennent des résultats semblables à l'humain [73], mais il est toujours utilisé comme référence pour évaluer de nouvelles méthodes.

3.4.3 ImageNet

ImageNet [23] est l'une des plus grandes bases de données de classification. Elle contient 1 million d'images appartenant à 1000 classes. Elle fait partie du défi ILSVRC qui, depuis 2010, pousse les chercheurs à se dépasser. Il est commun en vision par ordinateur d'évaluer son modèle sur cet ensemble et ensuite de distribuer les poids ainsi entraînés. Cela permet aux autres chercheurs d'initialiser leur entraînement à partir de ceux-ci. On peut aussi faire de la localisation sur cet ensemble, mais la classification est plus populaire. Dû au nombre élevé de classes, le protocole d'évaluation est différent des autres jeux de données. En effet, on préfère évaluer la *top-5 accuracy*

où une prédiction est bonne si elle est dans le top 5 des prédictions du modèle. Ce défi s'est terminé en 2017 où Squeeze-and-Excite [43] a été sacré gagnant avec 95.53% de *top-5 accuracy*.

3.4.4 CompCars

Le Comprehensive Cars dataset[109] ou CompCars est un jeu de données proposé lors de CVPR 2015. Il contient 136,726 images de 1,716 modèles de voiture différents. Dû au grand nombre de classes et de la grande variété des images, l'ensemble est jugé très difficile. En conséquence, les modèles modernes atteignent à peine les 50% de précision.

Une tâche possible sur cet ensemble, en plus de la classification, est la prédiction d'attributs. À partir de l'image d'une voiture, on demande de prédire le nombre de portes, la vitesse maximale, le type de voiture et le nombre de sièges.

On peut également utiliser cette base de données pour la comparaison de véhicules. Dans cette tâche, on demande à l'algorithme d'apprentissage si deux images sont celles d'un même modèle. On obtient des résultats intéressants où les algorithmes de l'état-de-l'art obtiennent plus de 75% de précision.

3.4.5 notMNIST

notMNIST[11] est une base de données semblable à MNIST, mais pour des lettres de A à J provenant de différentes polices de caractères. Cet ensemble est légèrement plus difficile que MNIST, mais les réseaux à convolution modernes obtiennent tout de même plus de 95% de justesse sans difficulté.

3.4.6 STL-10

STL-10[15] est une base de données semblable à CIFAR10. Elle est par contre beaucoup plus difficile dû au faible nombre d'exemples (500 par classe) et les images ont une différente résolution (96×96). Par contre, 100,000 images non annotées sont disponibles pour de l'entraînement non supervisé.

3.4.7 SVHN

Le Street-view house number dataset[77] ou SVHN est une base de données contenant des images de chiffres sur les panneaux d'adresses civiques. Les images de 32×32 pixels ont comme caractère principal un chiffre entre 0 et 9. Certaines images contiennent aussi les chiffres voisins, ce qui rend la tâche plus difficile. Il est jugé comme l'ensemble le plus difficile en classification de caractères.

3.4.8 Inria Person Dataset

Le Inria Person Dataset[22] est un ensemble de données dévolu à la détection de piétons. Il contient deux tâches : la classification et la localisation. Dû au grand nombre d'erreurs dans la tâche de localisation, on préfère l'utiliser pour faire de la classification. En effet, seuls les piétons de plus de 100 pixels de hauteur sont annotés et les boîtes ne sont pas très précises ce qui peut compliquer l'évaluation des méthodes. Pour la tâche de classification, on cherche à différencier les piétons de l'arrière-plan.

3.4.9 SeeFood

SeeFood[7] est un sous-ensemble du populaire jeu de données Food-101[9]. L'objectif est de classifier correctement les images de *hot-dogs* des autres types de plats. Son importance vient du fait qu'il existe peu d'ensembles d'images avec seulement deux classes. Bien que peu de méthodes aient été appliquées à cet ensemble, SeeFood est extrêmement difficile dû à son faible nombre d'exemples. En effet, on ne dispose que de 500 exemples par classe ce qui est problématique pour les méthodes modernes.

3.4.10 Pulmo-X

Pulmo-X[46] contient deux ensembles de radiographies pulmonaires provenant de deux hôpitaux. Les deux ensembles contiennent deux classes : healthy, not healthy. Les deux ensembles contiennent plus de 1500 sujets où plusieurs informations telles que le sexe, l'âge et le diagnostic sont présentes. Malgré le faible nombre d'exemples, les réseaux à convolution modernes sont en mesure d'approcher la perfection sur celui-ci.

Chapitre 4

Article : Spectral Metric for Dataset Complexity Assessment

Cet article présente notre nouvelle mesure de complexité, s'inspirant de la théorie des graphes, qui a été soumise à CVPR 2019.

Cette méthode propose de traiter l'estimation de complexité comme un problème de type *GraphCut* où il existe un coût à couper les arrête d'un graphe. En modélisant les relations entre classes comme un graphe, on peut déterminer la complexité de retirer toutes les arrêtes du graphe.

Le professeur Pierre-Marc Jodoin a eu l'idée initiale, travaillé à la conception des expériences. Il a aussi participé à la rédaction ainsi qu'à la revue de littérature. Andrew Aachkar a participé à la rédaction de l'article. Pour ma part, j'ai implémenté la mesure, j'ai eu l'idée d'utiliser des projections ainsi que d'utiliser le *eigengap* et j'ai effectué les expériences.

Nos contributions sont les suivantes :

— Nous proposons une nouvelle mesure de complexité robuste et en accord avec les performances obtenue par les réseaux de neurones.

— Nous proposons une méthode pour analyser visuellement les relations entre classes.

— Nous montrons que la base de données MioTCD contient trop d'éléments et peut être coupée de 20% sans affecter les performances.

Spectral Metric for Dataset Complexity Assessment

Frédéric Branchaud-Charron[1], Andrew Aachkar[2], Pierre-Marc Jodoin[1]

[1] *Département d'informatique, Université de Sherbrooke, Sherbrooke, Canada*

[2] *Miovision inc., Waterloo, Canada*

4.1 Abstract

In this paper, we propose a new measure to gauge the complexity of image classification problems. Given an annotated image dataset, our method computes a complexity measure called the cumulative spectral gradient (CSG) which strongly correlates with the test accuracy of convolutional neural networks (CNN). The CSG measure is derived from the probabilistic divergence between classes in a spectral clustering framework. We show that this metric correlates with the overall separability of the dataset and thus its inherent complexity. As will be shown, our metric can be used for dataset reduction, to assess which classes are more difficult to disentangle, and approximate the accuracy one could expect to get with a CNN. Results obtained on 11 datasets and three CNN models reveal that our method is more accurate and faster than previous complexity measures.

4.2 Introduction

The number of image-based datasets designed to train deep convolutional neural networks (CNN) have been on the rise in the past few years [11, 15, 53, 57, 66, 77, 109]. One reason for this is the indisputable efficiency of CNNs at classifying image data [14, 40, 53, 95].

A common challenge that arises when building a new image dataset for training a CNN is to identify how challenging the classification problem is, which classes are the most difficult to disentangle, and correspondingly what is the minimum dataset size required to train a CNN. As of today, there is no standard framework to make such determinations. The common way to assess the complexity of an image dataset is by training, finetuning and comparing results from several CNNs, the test accuracy

being the usual measure for complexity. However, this procedure is time consuming and, most importantly, requires a fully-annotated dataset which is not available when in the process of building it.

Unfortunately, one cannot predict the accuracy of a CNN by only looking at its architecture. As mentioned by Zhang *et al.* [112] in their attempt to understand why deep neural nets generalize well, they showed that deep neural networks can easily fit with zero training error on any input data, including pure random noise. This underlines the sole ability of CNNs to project any input data into a linearly separable space (and thus have a zero training error) while sometimes having poor generalization abilities. Their conclusion is that the structure of a neural net, its hyperparameters, its depth, and its optimizer cannot be used alone to predict its generalization capabilities.

Assessing the complexity of a classification problem may instead start from the analysis of the training set at hand with the goal of deriving useful complexity measures (c-measures) [3, 6, 26, 41, 97]. The goal of c-measures is to assess how entangled classes are assuming that datasets with overlapping classes are more difficult to analyze than those with well separated classes. C-measures have been shown effective for a number of applications such as classifier selection [10], automatic noise-filtering adjustment [92], dataset reduction [59], and hyperparameter tuning [70].

Unfortunately, existing c-measures have not been designed for large image datasets used to train deep neural networks. While some c-measures assume that classes are linearly separable in their original feature space [41], others work only for two-class problems [3, 6, 42]. Also, some c-measures are prohibitively slow and memory expensive as they require the analysis of matrices whose size is in the order to the number of training samples and/or the feature dimension size [6, 26].

Another important limitation with existing c-measures is the fact that they process raw input data. While this was shown valid for some classification problems [10], it is ill suited for deep neural nets since their learning procedure allows them to project input data onto a different and more easily separable space.

In this paper, we present a novel c-measure adapted to modern image classification problems. Instead of processing raw input data like previous approaches, our method first projects the input images onto a lower-dimensional latent space. This allows to

analyze data whose features are better adapted to what CNNs learn. Our method then estimates pairwise class overlap with a Monte-Carlo method which leads to an inter-class similarity matrix. Following the spectral clustering theory, we compute a $K \times K$ Laplacian matrix where K is the number of classes. Finally, the spectrum of this matrix is used to derived our cumulative spectral gradient (CSG) c-measure.

The main advantages of our proposed c-measure are as follows :

1. It naturally scales with the number of classes and the number of images in the dataset ;
2. Our metric is fast to compute and does not require the computation of prohibitively large matrices ;
3. It has no prior assumption on the distribution of the data ;
4. It gives a strong insight on which classes are easily separable and those that are entangled ;
5. The metric is highly correlated with CNN generalization capabilities ;
6. It can be easily used for dataset reduction.

4.3 Previous works

The goal of a c-measure is to characterize the difficulty of a classification problem. While several c-measures have been proposed in the past, those by Ho and Basu are by far the most widely used [41]. They proposed 12 different descriptors called F1, F2, F3, L1, L2, L3, N1, N2, N3, N4, T1, and T2. F1 is a Fisher's Discriminant Ratio, F2 measures the inter-class overlap, and F3 is the largest fraction of points one can correctly classify with a stump decision function. L1, L2 and L3 measures the linear separability of the data, while N1, N2, N3 and N4 are nearest neighbor measures which estimate the inter-class overlap. As for T1, it measures the total number of hyperspheres one can fit into the feature space of a class and T2 is the ratio between the total number of training samples N divided by the dimensionality of the data d.

While the c-measures by Ho and Basu have been shown effective for small non-image datasets [10], those metrics are less suited to analyze large and complex image datasets. For example, F1, F2, F3, L1, L2 and L3 assumes the data is linearly separable which is an over-simplistic assumption when considering modern image datasets. F1

requires the computation of $d \times d$ matrices which is problematic memory wise for large d (i.e. for medium to large images) and F3 measures the linear separability of each class by accounting for each feature independently which is prohibitively slow when both N and d are large. T1 is also prohibitively slow as N gets large since it requires to grow an hypersphere around each data point and T2 is not a good complexity predictor as will be shown in the results section.

Although Ho and Basu's metrics were designed for two-class problems, some researchers generalize it to more than two classes by averaging measures obtained between all possible pair of classes [81, 97]. Also, although recent generalizations of the Ho-Basu c-measures have been proposed [3, 18, 97], none addresses explicitly the problem of classifying large image datasets.

Other c-measures have been proposed. For example, Baumgartner and Somorjai [6] proposed a metric adapted to small biomedical datasets with high dimensionality data. Unfortunately, their c-measures are for two-class problems, assume that the data is linearly separable and require the decomposition of $N \times d$ matrices which is only tractable when N and d are small. Duin and Pekalska [26] quantify the complexity of a dataset with metrics derived from a dissimilarity matrix of size $N \times M$ where N is the training set size and M is the number of "representation" vectors randomly sampled from the training set. They report results on several datasets including two image datasets [1] which contains 2000 or less black and white digits. The authors used the Euclidean distance to measure the similarity between two images, a metric that does not generalize well to real-world images [106].

Like we do, some methods build a graph from the dataset to characterize the intra and inter-class relationships [28, 76]. This type of method requires building a $N \times N$ distance matrix which is problematic memory wise for large datasets. For example, the Hub score by [64] requires to compute $A^T A$ where A is a $N \times N$ adjacency matrix.

To our knowledge, Li *et al.* [60] are the only ones who proposed a c-measure applied specifically to modern image datasets and deep neural networks. They called their measure the Intrinsic Dimension which is the minimum number of neurons a model needs to reach its best performances. They show that adding more neurons past the Intrinsic Dimension does not improve test accuracy. Unfortunately, as opposed to

1. https ://www.nist.gov/srd/nist-special-database-19

what we seek to do, their measure requires multiple training of image classification CNNs through a grid-search approach which is slow and tedious. More details on c-measures can be found in the recent survey paper by [64].

4.4 Proposed Method

4.4.1 Class overlap

At the core of our c-measure is the notion of class overlap. Let x be an input image and $\phi(x) \in I\!R^d$ an embedding for that image. As will be discussed later, ϕ can be any function that projects x to a new dimensional space where images with similar content are close together and the other ones further away. The overlap between two classes C_i and C_j refers to the overall area in the feature space for which $P(\phi(x_k)|C_i) > P(\phi(x_k)|C_j)$ when $\phi(x_k)$ is a member of class C_j. Class overlap can thus be formulated as [79] :

$$\int_{I\!R^d} \min\left(P(\phi(x)|C_i), P(\phi(x)|C_j)\right) d\phi(x). \tag{4.1}$$

Unfortunately, the direct calculation of this integral is prohibitively complicated for non-parametric distributions and when d (the dimensionality of the embedded space) is large. Since the overlap between two classes is related to the similarity of their distributions, one may instead use a probability distribution distance function such as the Kullback-Leibler divergence or the Kolmogorov-Smirnov test as a surrogate for Eq.(4.1). One such function is the probability product kernel of Jebara *et al.* [47] :

$$\int_{I\!R^d} P(\phi(x)|C_i)^\rho P(\phi(x)|C_j)^\rho d\phi(x) \tag{4.2}$$

which is a generalization of the Bhattacharyya kernel (and the Hellinger distance) when $\rho = 1/2$. While computing Eq.(4.2) is as complex as computing Eq.(4.1) for an arbitrary value of ρ, simplification occurs when $\rho = 1$. In that case, the kernel becomes the inner-product between the two distributions $\int_{I\!R^d} P(\phi(x)|C_i)P(\phi(x)|C_j)d\phi(x)$ which is the expectation of one distribution under the other : $E_{P(\phi(x)|C_i)}[P(\phi(x)|C_j)]$ or $E_{P(\phi(x)|C_j)}[P(\phi(x)|C_i)]$.

Formulating the inter-class divergence as an expectation function allows one to use Monte-Carlo to approximate it :

$$E_{P(\phi(x)|C_i)}[P(\phi(x)|C_j)] = \frac{1}{M}\sum_{m=1}^{M} P(\phi(x_m)|C_j) \tag{4.3}$$

where $\{\phi(x_1), ..., \phi(x_M)\}$ are M samples i.i.d. from $P(\phi(x)|C_i)$. One can thus approximate the divergences between two class distributions by averaging the probability of M samples of class C_i to be in class C_j or vice versa. Computing inter-class divergences leads to a $K \times K$ similarity matrix $\mathcal{S}$ where K is the total number of classes and $\mathcal{S}_{ij}$ is the Monte-Carlo approximation of the divergence between C_i and C_j.

Since the underlying model of $P(\phi(x_m)|C_j)$ is a priori unknown, we approximate it with a K-nearest estimator :

$$p(\phi(x) \mid C_j) = \frac{K_{C_j}}{MV} \tag{4.4}$$

where V is the volume of the hypercube surrounding the k closest samples to $\phi(x)$ in class C_j, M is the total number of samples selected in class C_j and K_{C_j} is the number of neighbors around $\phi(x)$ of class C_j. The rationale behind our method is as follow : the complexity of a classification problem is affected by several issues such as ambiguity, noise in the annotations or non discriminative data features. We estimate this noise with our method using multiple weak classifiers performing noise estimation.

4.4.2 Spectral Clustering

The $K \times K$ similarity matrix $\mathcal{S}$ embodies the overall complexity of a dataset by means of class overlap. Our goal is to extract a measure from $\mathcal{S}$ that would summarize the complexity of that dataset. For that, we rely on the spectral clustering theory [105] that we briefly review in this section.

Let G be an undirected similarity graph $G = (V, E)$ where V is a set of nodes connected by edges E. An edge E_{ij} is an arc connecting two nodes i and j and whose weight $w_{ij} \geq 0$ encodes how close these two nodes are. A weight of 0 implies no connection between i and j whereas a large weight implies strong similarity (in our

case, a weight of 1 implies that i and j are identical). The weight of all edges are put in a $n \times n$ adjacency matrix W where n is the total number of nodes. Note that W is symmetric and positive semi-definite due to the undirected nature of the graph which implies that $w_{ij} = w_{ji}$.

The goal of spectral clustering is to partition G into subgraphs such that the edges between the subgraphs have minimum weight. A set of subgraphs $\{G_1, ..., G_l\}$ is valid when $G_i \cap G_j = \emptyset$, $\forall i \neq j$ and $G_1 \cup ... \cup G_l = G$. An optimal partition of G is one for which the cut has minimum cost : $costCut(G_1, ..., G_l) = \sum w_{ij}$ for i and j in different subgraphs.

Spectral clustering provides an elegant framework to recover the subgraphs with minimum cut. It starts with a Laplacian matrix whose simplest form is $L = D - W$ where D is a degree matrix $D_i = \sum_j w_{i,j}$. L is symmetric and positive semi-definite, it contains n eigenvalues $\{\lambda_0, ..., \lambda_{n-1}\}$ that are real and non negative with $\lambda_0 = 0$ and $\lambda_{i+1} \geq \lambda_i$. This set of eigenvalues is called the *spectrum* of L. Interestingly, the n eigenvectors associated to the eigenvalues can be seen as indicator vectors that one can use to cut the graph. Also, the magnitude of their associated eigenvalues is related to the cost of their cut [75]. As such, the eigenvectors associated to the lowest eigenvalues are those associated to the partitions of minimum cost.

4.4.3 Inter-class adjacency matrix

We formulate our c-measure within the spectral clustering framework for which each node is a class index. In our case, W and L are $K \times K$ matrices where K is the total number of classes. As such, the weight $w_{i,j}$ is the distance between the likelihood distributions of classes i and j. Thus, a simple dataset for which each pair of classes has little overlap would produce a sparse Laplacian matrix L whose spectrum contains small eigenvalues. On the other hand, a more complex dataset with stronger class overlap would lead to a spectrum with larger eigenvalues.

Since the similarity matrix $\mathcal{S}$ was obtained with a Monte-Carlo approximation of the Jebara kernel, it is not symmetric and thus cannot be used as an adjacency matrix W. Instead, we consider each column $\mathcal{S}_i$ as a signature vector of each class i so two classes with similar likelihood distributions would also have a similar signature

vector $\mathcal{S}_i$ and vice versa. The Bray-Curtis distance function [34] is used in biology to compute the dissimilarity between two sites. In our cases, each class distribution is considered a sampling site :

$$w_{ij} = 1 - \frac{\sum_k^K |\mathcal{S}_{ik} - \mathcal{S}_{jk}|}{\sum_k^K |\mathcal{S}_{ik} + \mathcal{S}_{jk}|}. \tag{4.5}$$

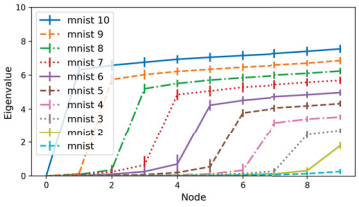
This equation implies that $w_{ij} = 0$ when the distributions of classes i and j do not overlap and $w_{ij} = 1$ when the distributions are identical.

4.4.4 Runtime improvement

As will be shown later, current techniques are time consuming and as such, our technique propose interesting runtime improvement using Monte-Carlo estimation. In practice, these improvements are useful when building new datasets. Annotating a new dataset is time consuming and would benefit from better methodologies. For instance, driving the annotation effort toward high complexity classes would be more useful than annotating already low complexity classes. To perform this, the dataset needs to be rapidly iterated upon to keep track of the high complexity classes. This requires that the complexity measure is fast to compute. Computing the adjacency matrix W with the Bray-Curtis function as well as the Monte-Carlo method (Eq.(4.3)) is 40 times faster than with a naive implementation (Eq.(4.2)) for a K=10 class problem. This explains why our method is fast and gets good results even with a small number of samples. We came to that number as follows.

First, let us mention that the most computationally intensive operation is the point-wise estimation of a probabilistic distribution function (pdf) $P(\phi(x)|C)$. Since computing Eq.(4.2) requires M estimations of $P(\phi(x)|C_j)$, the $K \times K$ similarity matrix S requires a total of $K^2 \times M$ pdf estimations, where K is the number of classes and M the number of samples. Also, since Eq.(4.5) requires no additional pdf estimation, our method requires a total of $K^2 \times M$ pdf estimations to compute the adjacency matrix W.

However, since the Bray-Curtis distance function combines two R^K vectors S_i and S_j, it incorporates the statistics of $2 \times K \times M$ samples at each entry w_{ij} of W. If the



Dataset	CSG	E.R.
mnist 10	5.51	0.91
mnist 9	5.04	0.78
mnist 8	4.53	0.69
mnist 7	3.79	0.61
mnist 6	3.31	0.51
mnist 5	2.70	0.39
mnist 4	2.16	0.30
mnist 3	1.52	0.18
mnist 2	0.84	0.13
mnist	0.12	0.01

figure 4.1 – [Left] Spectrum of ten noisy versions of MNIST and [right] our CSG c-measure with the error rate (E.R.) of an AlexNet CNN (figure best viewed in color).

same number of samples were to be used by a naive implementation, i.e. that w_{ij} was to be computed with $2 \times K \times M$ samples and Eq.(4.2), the computation of W would require a total of $4 \times K^3 \times M$ pdf estimations, i.e. $4 \times K$ more pdf estimations than for our method. From there we conclude that our method is 40 times faster than a naive implementation when $K = 10$. Please note that this is in line with empirical results.

4.4.5 The CSG complexity measure

As mentioned before, a dataset with a low eigenvalue spectrum indicates a low inter-class overlap and thus easily separable classes. To illustrate this, we put in Fig. 4.1 the spectrum of the MNIST dataset (the bottommost cyan curve) which we obtained by processing raw images. Since MNIST contains 10 classes, its spectrum contains 10 eigenvalues. Being a simple dataset, MNIST's spectrum contains mostly near-zero values. We then randomly swap elements between classes to force their distribution to strongly overlap, making this noisy version of MNIST more complex. We first swap elements between two classes (MNIST 2), then between three classes (MNIST 3) all the way to 10 classes (MNIST 10). As one can see, these noisy versions of MNIST lead to a gentle progression of the spectrum profiles. The more entangled classes are, the larger the eigenvalues are. Also, the sooner a strong spectrum gradient occurs ($\lambda_{i+1} - \lambda_i$) the more difficult the dataset is (this gradient discontinuity is also called the eigengap in the spectral clustering literature [105]).

The overall complexity of the dataset is thus related to the area under the spectrum curve as well as the position of the eigengap. To account for both observations, we

first normalize the eigengap by its horizontal position :

$$\Delta\widetilde{\lambda_i} = \frac{\lambda_{i+1} - \lambda_i}{K - i}. \tag{4.6}$$

The normalization by $K - i$ is at the core of our metric. Depending on where the largest eigengap occurs, its maximum value can only be of $K - i$. The difficulty of cutting the graph is thus related to the position of the largest eigengap. Our c-measure (the cumulative spectral gradient (CSG)) is the cumulative maximum (cummax) of the normalized eigengaps :

$$CSG = \sum_i \text{cummax}(\Delta\widetilde{\lambda})_i. \tag{4.7}$$

With a cummax, between two spectrums with the same area under the curve, our CSG measure will be larger for the one with the left-most eigengap. The CSG values for the noisy MNIST datasets are shown on the right of Fig. 4.1 along side with the test error rate obtained with an AlexNet CNN [54]. As can be seen, our CSG c-measure is heavily correlated to the complexity of the datasets.

Our method is summarized in Algorithm 3.

Data: Dataset=$\{(\phi(x_1), t_1), ..., (\phi(x_N), t_N)\}$
Args: M, k
Result: CSG score
Compute inter-class similarity matrix $\mathcal{S}$ with Eq.(4.3) and (4.4) $\forall$ pair of classes C_i, C_j
Compute W (Eq.(4.5))
$L \leftarrow D - W$
$\{\lambda_1, ..., \lambda_K\} \leftarrow \text{EigenValues}(L)$
Compute CSG (Eq.(4.7))
return CSG

Algorithm 3: The CSG c-measure algorithm.

4.5 Results

4.5.1 Embeddings

As mentioned before, the input images x are projected to an embedding space with a function $\phi(x)$. In this paper, we tested four projection functions :

1. Raw ; the identity function $\phi(x) = x$;
2. t-SNE ; the t-SNE function [103] which projects the raw input images down to a 2D space ;
3. CNN_{AE} ; the embedding of a 9-layer CNN-Autoencoder trained for 100 epochs ;
4. CNN_{AE} t-SNE ; the t-SNE function applied to the embedding of the CNN-autoencoder.

4.5.2 Datasets

In order to gauge performance of our method, we used several image classification datasets of various difficulty levels. Of those datasets, six contain 10 classes, one contains 11 classes and three contain two classes. These datasets are summarized in Table 4.1 and sorted according to the test error rate (E.R.) obtained with an AlexNet CNN [54]. Note that we replaced the AlexNet local response norm with a batch-norm [45], trained it for 500 epochs on each dataset with a batch size of 32 and the SGD optimizer with the same parameters than in the original paper but without data augmentation. We used Keras [13], Tensorflow [1] and an Nvidia Titan X GPU.

The datasets are the well-known MNIST [57] and CIFAR10 [53]. There is also notMNIST [11], a synthetic dataset of 18,724 letters made of unconventional fonts, and the Street View House Numbers (SVHN) dataset [77], one of the most challenging digit classification datasets with 73,257 images of low resolution street numbers. We also use MioTCD [66], a large dataset of 648,959 vehicles pictured by traffic cameras with varying orientation angles, resolution, time of the day and weather conditions. STL-10 [15] is a 10-class dataset similar to CIFAR-10 but with larger images (96×96 instead of 32×32) and fewer training samples (5,000 instead of 50,000). SeeFood [7] is a two-class dataset (Hot-dog vs No Hot-dog) with 498 samples derived from the Food-101 dataset [9]. We also use the well-known Inria pedestrian dataset [22] containing

Datasets	E.R.	K	N	Content
MNIST	0.01	10	50k	Hand written digits
MIO-TCD	0.03	11	649k	Traffic images
notMNIST	0.05	10	18.7k	Printed digits
SVHN	0.08	10	73.3k	Printed digits
Inria	0.10	2	3.6k	Pedestrians
CIFAR10	0.12	10	50k	Various real images
Pulmo-X	0.23	2	662	Pulmonary X-Rays
SeeFood	0.38	2	500	Images of food
STL-10	0.68	10	5k	Various real images
CompCars	0.70	10	6k	Pictures of cars

tableau 4.1 – Datasets used to validate our method with the test error rate (E.R.) obtained with an AlexNet CNN [54], the number of classes K, the training set size N and a short summary.

38,634 RGB images of pedestrians or not, and Pulmo-X [46], a two-class pulmonary chest X-Ray dataset for tuberculosis detection containing 662 images. Finally, CompCars [109] is a dataset containing 1,716 car categories of different makes and models. For our experiments, we selected the 10 makes with the highest count and resized the images to 128×128, giving us 500 samples per class.

We followed the evaluation methodology specific to each dataset, i.e. we trained and tested the methods on the training and testing set provided with the datasets. However, there were two datasets without pre-determined train/test split : notMNIST and Pulmo-X. For those two datasets, we made a 80-20 Train/Test split and kept the same class proportion.

Hyper-parameters

Our algorithm has two main hyper-parameters : M the number of samples per class used by the Monte Carlo method in Eq.(4.3) and k the number of neighbors used to compute the likelihood distribution of each class in Eq.(4.4). In Table 4.2, we show the Pearson correlation score between our c-measure with the $\underline{\text{CNN}_{AE}\text{ t-SNE}}$ embedding and the error rate of AlexNet on the six 10-class datasets (upper table) as well as the average processing time of our Algo 1 (lower table). As one can see, the choice for k and M has little impact on the quality of the results (except for

		k						
		1	**3**	**5**	**7**	**9**	**11**	
M	**2**	0.81	0.79	0.80	0.75	0.76	0.73	Pearson Corr.
	50	0.97	0.97	0.97	0.96	0.96	0.97	
	100	0.97	0.97	0.98	0.98	0.97	0.97	
	200	0.98	0.98	0.98	0.98	0.98	0.98	
	300	0.97	0.97	0.97	0.98	0.97	0.98	
	400	0.97	0.97	0.97	0.97	0.97	0.97	

M	**2**	0.02	0.02	0.02	0.02	0.02	0.02	Timing (s)
	50	0.30	0.30	0.29	0.30	0.29	0.27	
	100	0.60	0.61	0.60	0.60	0.60	0.60	
	200	1.22	1.21	1.23	1.19	1.20	1.22	
	300	1.82	1.82	1.82	1.78	1.83	1.79	
	400	2.42	2.38	2.42	2.41	2.39	2.39	

tableau 4.2 – Correlation values [upper table] and average processing times of Algo 1 in seconds [lower table] for various combinations of hyperparameters M and k.

when M is very small). Also, while the runtime scales almost linearly with M, our method is still fast with timings below 3 seconds, even with $M = 400$ samples per class. This shows that our method does not require a careful adjustment of its hyper-parameters. We found this as well for the other embeddings we tested. As such, we will use $M = 100$ and $k = 3$ for the remainder of this section.

4.5.3 Experimental results

Comparison with other c-measures

We compared our method to the most widely implemented c-measures, i.e. those by Ho and Basu [41]. We used the C++ DCol library provided by the authors [81] and processed the six 10-class datasets. We thus followed the original methodology provided by the authors which implies no embedding. In addition, we tested two other metrics derived from the spectral theory : the maximum eigenvalue ($\max \Lambda$) and the area under the curve (AUC). These methods are known in the literature as being related to the similarity between nodes [96]. These turn out to perform worse than our CSG metric. Results are reported in Table 4.3 together with our method with

c-measure	Corr.	p-value	Time (s)
N4	0.141	.790	3,744
F3	0.290	.577	3,924
F1	0.483	.332	72
F2	0.366	.476	72
T1	0.642	.170	36,108
T2	0.655	.158	72
N2	0.677	.140	36,180
F4	0.760	.079	3,644
N1	0.767	.075	17,748
N3	0.803	.054	36,216
$\max_i \lambda_i$ CNN_{AE} t-SNE	0.88	0.02	0.3 (18,900)
$\sum_i \lambda_i$ CNN_{AE} t-SNE	0.94	$\leq$**0.01**	0.3 (18,900)
CSG Raw	0.696	.125	50 (*NA*)
CSG CNN_{AE}	0.823	.044	3.6 (13,300)
CSG t-SNE	0.903	.014	0.7 (6,084)
CSG CNN_{AE} t-SNE	**0.968**	$\leq$**0.01**	0.3 (18,900)

tableau 4.3 – Correlation between the accuracy of AlexNet and 10 c-measures by Ho-Basu [41] and ours methods with four embeddings, the associated p-value and processing time (measured on CIFAR10).

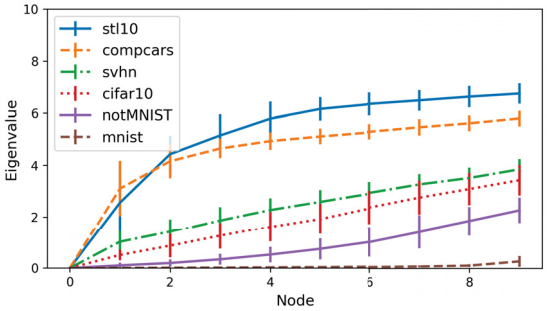


figure 4.2 – Laplacian spectrum for the 10-class datasets.

four embeddings.

The first column contains the Pearson correlation score between the error rate by an AlexNet CNN and each c-measure. As one can see, our method with the $\underline{\text{CNN}_{AE}}$, $\underline{\text{t-SNE}}$ and $\underline{\text{CNN}_{AE}\ \text{t-SNE}}$ embeddings have a better correlation than any of the existing c-measures with a p-value below the 0.05 bar. The best embedding is CNN_{AE} t-SNE with a significance p-value below 0.01. To illustrate how this embedding correlates with the dataset complexity, we put in Fig. 4.2 its Laplacian spectrum for the six 10-class datasets. As can be seen, the spectrum plots grow smoothly from the simplest dataset (MNIST), to slightly more complex datasets (notMNIST, CIFAR10 and SVHN) all the way to the most complex datasets (STL-10 and CompCars). Note that we will use the CNN_{AE} t-SNE embedding for the remaining of this section.

As for processing time, our method is faster than the best c-measures F4, N1 and N3. Note the value on the left is the time to execute Algo. 3 whereas the value in parenthesis is the processing time to train a CNN_{AE} and/or run t-SNE. Although that processing time is large (more than one hour) it is much faster than the previous best method N3. [2]

In Table 4.4, we provide our CSG c-measure with the test error rate of three CNN models as well as their Pearson correlation and p-value. As can be seen, our c-measure correlates well not only with AlexNet, but also with more recent ResNet-50 [40] and XceptionNet [14]. Also, our correlation and p-value with CNN error rates is significantly better than the best existing c-measures [41].

2. The timings were computed on CIFAR10 using a Intel®Xeon®CPU E5-1620 and a NVIDIA TITAN X.

		Error rate		
Datasets	CSG	AlexNet	ResNet-50	Xception
CompCars	2.93	0.70	0.88	0.86
STL-10	3.07	0.69	0.63	0.69
CIFAR10	1.00	0.18	0.19	0.06
SVHN	1.15	0.08	0.07	0.03
notMNIST	0.72	0.05	0.04	0.03
MNIST	0.11	0.01	0.05	0.01
N3 Ho-Basu/CNN Corr		0.803	0.727	0.681
N3 Ho-Basu/CNN p-val		0.054	0.102	0.136
CSG/CNN t-SNE Corr		**0.968**	**0.935**	**0.951**
CSG/CNN t-SNE p-val		**0.01**	**0.006**	**0.003**

tableau 4.4 – [Top] CSG c-measure alongside with test error rates for 3 CNN models on six datasets.[Bottom] Pearson correlation and p-value between our method and CNN and between the N3 c-measure [41] and CNN.

We also tested our method on two-class image classification problems. We used the Inria, SeeFood, and PulmoX datasets as well as the deer-dog subset of CIFAR10. Results reported in Table 4.5 show that our method correlates well with the CNN models, especially AlexNet. Our correlation scores are also better than those of the best c-measure of Ho-Basu (although by a slight margin) although it was specifically designed for two-class problems.

Datasets	CSG	AlexNet	ResNet-50	Xception
SeeFood	0.95	0.38	0.34	0.21
PulmoX	0.55	0.23	0.16	0.11
deer-dog	0.39	0.20	0.02	0.02
Inria	0.32	0.10	0.07	0.03
N3 Ho-Basu/CNN Corr		0.976	0.852	0.862
N3 Ho-Basu/CNN p-val		0.01	0.148	0.138
CSG/CNN Corr		**0.995**	**0.860**	**0.887**
CSG/CNN p-val		**0.006**	**0.130**	**0.113**

tableau 4.5 – [Top] CSG c-measure alongside with test error rates for 3 CNN models on six datasets.[Bottom] Pearson correlation and p-value between ours methods and CNN and between the N3 c-measure [41] and CNN.

Dataset reduction

Dataset reduction (also known as instance selection [59]) consists in reducing as much as possible the number of elements in a dataset without losing trained CNN accuracy. One way of doing so is by iteratively removing elements from the dataset up to a point where the CSG measure increases sharply.

We first tested our method on the MIO-TCD dataset [66], a large dataset used for a 2017 CVPR challenge and for which CNN methods got accuracies of up to 98%. Such high accuracies suggest that the dataset is overcomplete and could be reduced without affecting much the CNN accuracy. Results for various reduction ratios are shown in Figure 4.3. As one can see, the CSG (red dots) stays roughly unchanged for reduction ratios below 80% but then increases sharply after that. This is inline with the AlexNet test error rate (blue line) although it took less than 5 minutes to produce the CSG measures and 5 days the AlexNet results. We used the same CNN_{AE} embedding for all ratios. We got a Pearson correlation of 0.956 between our CSG dots and the error rate values shown in Figure 4.3.

Dataset reduction can also be used to measure the similarity between two datasets with very different sizes like CIFAR10 (5,000 training samples per class) and STL-10 (500 training samples per class). While these datasets have visually similar content, the CNN error rates on it are very different (see Table 4.1). To measure the true distance between those datasets, we progressively reduced the number of samples for each CIFAR10 class to reach that of STL-10. The results in Table 4.6 show the close bound between our metric and the number of samples in the dataset. With only 500 samples, CIFAR10 gets a CSG score and a CNN accuracy similar but not identical to that of STL-10. This shows that the datasets are similar but not identical, probably due to the fact that the CIFAR10 Frog class has been replaced by a Monkey class in STL-10 (supplementary material). Here again, it took roughly one minute to produce the CSG scores (after having trained the embedding) and 4 days for the CNN error rates.

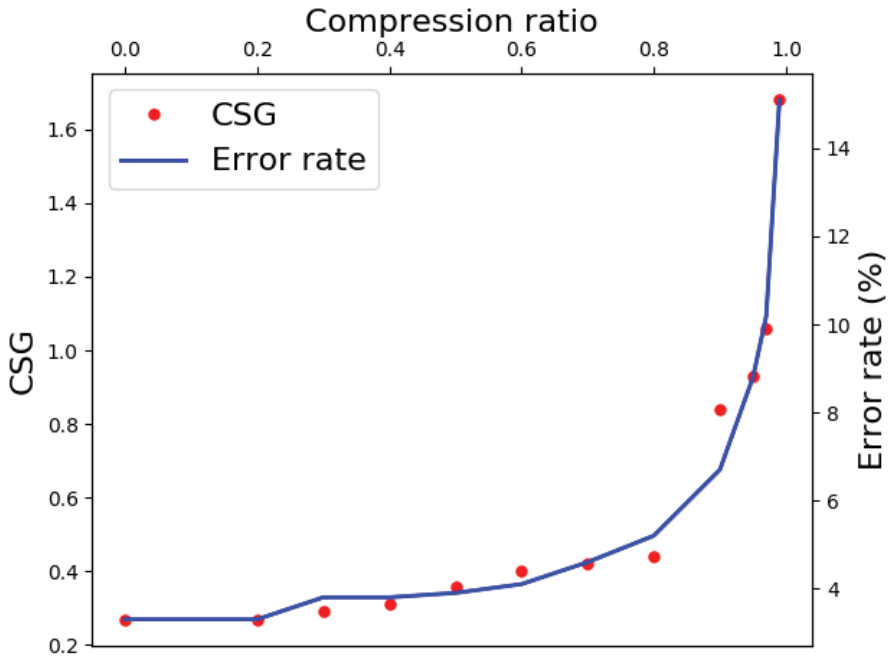


figure 4.3 – Our c-measure and AlexNet accuracy obtained while reducing the size of the MioTCD dataset.

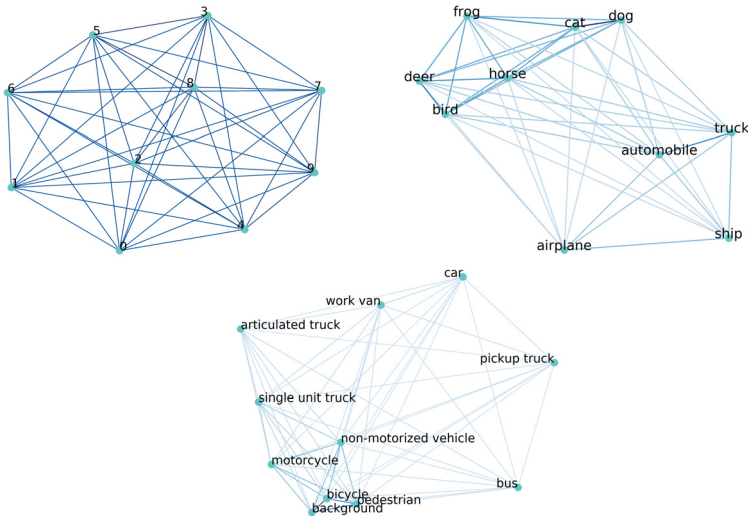


figure 4.4 – 2D plots of our W matrix for MNIST, CIFAR10 and MioTCD.

figure 4.5 – [Top] our W matrix and [Bottom] AlexNet's confusion matrix for CIFAR10.

4.5.4 Confusion matrix

While our c-measure can gauge the overall complexity of a dataset with a single measure that correlates with CNN accuracies, we can also use the similarity matrix W (Eq.(4.5)) to analyze the inter-class distances. As such, one can use a dissimilarity matrix $S = 1 - W$ to visualize the dataset in 2D via an algorithm such as multidimensional scaling (MDS) [8]. This results into plots such as those in Fig. 4.4. While the classes of MNIST are all well separated, the CIFAR10 plots show that the cat and dog class as well as deer and bird are close to each other, probably due to similar contexts. As for MioTCD, the bicycle, motorcycle and pedestrian classes are in the same vicinity, mainly because of their small image resolution, they often contain more compression artifacts and hence be less feature-rich than other classes, making them confusing with featureless background.

As shown in Fig. 4.5, the W matrix strongly correlates to a real confusion matrix (here AlexNet). Here again, cat and dog as well as deer and bird are easily confused.

4.6 Conclusion

In this work, we proposed a novel complexity measure designed for image classification problems called the cumulative spectral gradient (CSG) which is more accurate and faster than previous methods. We showed that our metric has many uses such as instance selection and class disentanglement. We also showed that the CSG clo-

Dataset	CSG	Error rate
CIFAR10	1.10	0.18
CIFAR10 reduced=4500	1.10	0.19
CIFAR10 reduced=3500	1.26	0.20
CIFAR10 reduced=2500	1.44	0.24
CIFAR10 reduced=1500	2.16	0.28
CIFAR10 reduced=500	2.59	0.42
STL-10	3.16	0.68

tableau 4.6 – Effect of reducing the number of samples per class for CIFAR10 on our CSG metric and the AlexNet test error rate.

sely matches the accuracy achievable by standard CNN architectures, an important feature when assessing an image dataset.

A future direction of our research would be to determine a procedure to compare the relative complexity of classification problems with different number of classes. The analysis of random subsets of classes could be used as a common representation. Another important direction would be to generalize our method to segmentation and localization problems. As of now, it is not clear how these problems can be described by spectral clustering.

Another future work would be to incorporate our similarity matrix W in the optimization process of a neural network to minimize the interclass divergence. It is our intuition that the a priori knowledge of the interclass overlap could be used to force the optimizer to further separate entangled classes, a bit like the triplet loss does. Finally, our metric is not restricted to image classification datasets and could be used in other areas of machine learning such as speech recognition and natural language processing (NLP). These fields already use state-of-the-art embeddings such as Word2Vec [72] and would thus naturally fall into our CSG framework.

in a similar fashion as the triplet loss.

Supplementary Material

4.7 Instance selection

In the results section, we showed that STL-10 was a bit harder than CIFAR10 with 500 samples per class even though these datasets are visually very similar. To further our point, we show in Fig. 4.6 the MDS representation of both datasets. While the car-truck distance is smaller in CIFAR10, the six animal classes in STL-10 are overall closer together and thus a slightly more difficult to disentangle.

4.8 Comparison with other graph-based methods

We mentioned in the Previous works section that other graph-based methods have been proposed in the past. One could wonder how our measure differs from those. The main advantage of our approach compared to other graph-based methods is the fact that our graph embeds classes and not samples, thanks to the spectral clustering formalism. This leads to a $K \times K$ Laplacian matrix which is order of magnitude smaller that the $N \times N$ similarity matrix often required by other methods. This brings a huge advantage both memory and processing wise while allowing our method to naturally expand to the number of classes.

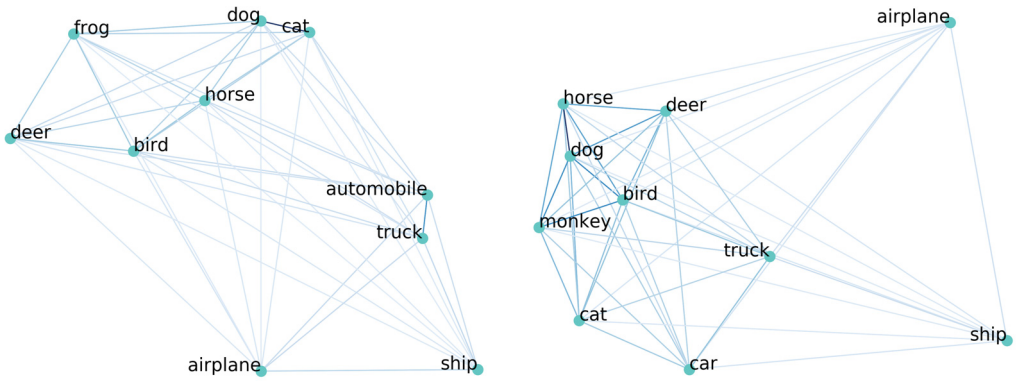


figure 4.6 – Comparison between the [Top] 2D plot of CIFAR10 with 500 samples per class and [Bottom] 2D plot of STL-10. As we see, the nodes in STL-10 seems closer than in CIFAR10.

Conclusion

Dans ce mémoire, nous avons exploré les applications de l'apprentissage profond, notamment la localisation, dans le domaine de l'analyse de trafic routier. En particulier, nous avons proposé une nouvelle base de données MioTCD pour évaluer les méthodes d'apprentissage profond sur une tâche contenant des environnements plus variés et plus difficiles que ce qui s'est fait auparavant. Par la suite, nous avons pu analyser les caractéristiques de l'ensemble de classification ce qui nous a permis de déterminer que l'ensemble était sur complet. De plus, nous avons proposé une nouvelle mesure de complexité adaptée aux techniques modernes qui fait usage de la théorie des graphes pour déterminer la complexité d'une base de données. Finalement, nous avons proposé une nouvelle méthode pour approximer l'orientation des objets lorsqu'on ne peut pas utiliser les algorithmes de flux optiques.

Plusieurs points restent à explorer dans ces domaines. En particulier, le domaine de l'analyse de complexité s'est pour l'instant limité à la classification. Un projet intéressant serait de le faire progresser vers d'autres tâches similaires telles que la segmentation et la localisation. Deuxièmement, les algorithmes de localisation ont de la difficulté dans les scènes fortement achalandées. Cela est souvent dû à l'algorithme de suppression des non-maximums qui fusionne les boîtes similaires. Une alternative serait de se tourner vers la segmentation d'instances, qui a rarement besoin de cet algorithme. Malheureusement, les modèles de segmentation d'instances ont un temps d'inférence important ce qui les rend inutilisables dans un contexte de localisation en temps réel. Finalement, on aimerait utiliser les mesures de complexité pour concevoir des modèles qui tiennent compte des difficultés de la base de données. Une avenue intéressante serait d'utiliser les mesures de complexité pour faire de la classification hiérarchique. Dans ce contexte, on obtient un arbre de décision qui nous informe sur

les parties du réseau de neurones à exécuter. Par exemple, nous n'avons pas besoin d'exécuter tout le réseau lorsqu'on traite une classe jugée facile.

En conclusion, bien que les méthodes d'apprentissage profond soient maintenant omniprésentes en vision par ordinateur, il reste beaucoup à faire pour que celles-ci utilisent les caractéristiques des bases de données à leur plein potentiel.

Annexe A

MIO-TCD : A new benchmark dataset for vehicle classification and localization

L'article suivant décrit la base de données MioTCD qui fut utilisée lors du *workshop* TSWC à la conférence internationale CVPR 2017. Elle a pour but d'évaluer les techniques modernes de localisation et de classification de véhicules. La base de données contient des scènes plus diversifiées que ses prédécesseurs. En effet, celles-ci offrent une plus grande variété de lieux, météo, types de véhicules ainsi que des artéfacts de compression comme on en voit dans la vie réelle.

Les employés de Miovision (Justin Eichel, Andrew Aachkar, Akshaya Mishra) ont pourvu les données, le système d'annotation et participé à la rédaction. Le professeur Pierre-Marc Jodoin et Zhiming Luo ont participé à l'élaboration du défi ainsi qu'au protocole des expériences. Zhiming Luo a évalué les méthodes de localisation et analysé les résultats. Carl Lemaire a participé à l'évaluation des techniques de classification. Pour ma part, j'ai participé à l'évaluation du système d'annotation, implémenter et évaluer plusieurs techniques de classification, confirmer les scores de YOLOV1-V2 et analyser les résultats de toutes les techniques de localisation selon le protocole de COCO.

Nos contributions sont les suivantes :

— Deux nouvelles bases de données de classification et de localisation.
— L'évaluation de plusieurs techniques modernes sur ces ensembles.
— Une analyse des difficultés de chaque technique selon les caractéristiques des objets à localiser.

Cet article a été publié dans le journal *IEEE Transactions on Image Processing*.

MIO-TCD : A new benchmark dataset for vehicle classification and localization

Zhiming Luo[1], Frédéric Branchaud-Charron[2], Carl Lemaire[2], Janusz Konrad[3], Shaozi Li[4], Akshaya Mishra[5], Andrew Aachkar[6], Justin Eichel[6], Pierre-Marc Jodoin[2]

[1] *Center of Information and Communication Engineering, Xiamen University, Xiamen, China*

[2] *Département d'informatique, Université de Sherbrooke, Sherbrooke, Canada*

[3] *Department of Electrical and Computer Engineering, Boston University, Boston, USA*

[4] *Cognitive Science Department, Xiamen University, Xiamen, China*

[5] *Systems Design Engineering Department, University of Waterloo, Waterloo, Canada*

[6] *Miovision inc., Waterloo, Canada*

A.1 Abstract

The ability to train on a large dataset of labeled samples is critical to the success of deep learning in many domains. In this paper, we focus on motor vehicle classification and localization from a single video frame and introduce the "MIOvision Traffic Camera Dataset" (MIO-TCD) in this context. MIO-TCD is the largest dataset for motorized traffic analysis to date. It includes 11 traffic object classes such as cars, trucks, buses, motorcycles, bicycles, pedestrians. It contains 786,702 annotated images acquired at different times of the day and different periods of the year by hundreds of traffic surveillance cameras deployed across Canada and the United States. The dataset consists of two parts : a "localization dataset", containing 137,743 full video frames with bounding boxes around traffic objects, and a "classification dataset", containing 648,959 crops of traffic objects from the 11 classes. We also report results from the 2017 CVPR MIO-TCD Challenge, that leveraged this dataset, and compare them with results for state-of-the-art deep learning architectures. These results demonstrate the viability of deep learning methods for vehicle localization and classification from a single video frame in real-life traffic scenarios. The top-performing methods achieve both accuracy and Kappa score above 96% on the classification dataset and

mean-average precision of 77% on the localization dataset. We also identify scenarios in which state-of-the-art methods still fail and we suggest avenues to address these challenges. Both the dataset and detailed results are publicly available on-line [67].

A.2 Introduction

The localization and classification of vehicles in the field of view of a traffic camera is the very first step in most traffic surveillance systems (e.g., car counting, activity recognition, anomaly detection, tracking, post-event forensics). Although subsequent processing may be different in each case, one often has to start by localizing foreground objects and then identifying what these objects are (trucks, cars, bicycles, pedestrians, etc.).

To the best of our knowledge, except for pedestrian applications, most traffic monitoring systems rely on motion features such as optical flow, motion detection, and vehicle tracking [108, 113, 62]. Motion features are then used to count the number of vehicles on the road, estimate traffic speed, and recover global motion trajectories. But these traffic monitoring systems all share a fundamental limitation : in order to function properly, they need high frame-rate videos so motion features can be reliably extracted [49]. Also, reliable object tracking is still a challenge for cluttered scenes, especially when vehicles are partly occluded [65].

Unfortunately, low frame-rate videos are common as many large-scale camera networks cannot stream and store high frame-rate videos gathered by thousands of cameras. Instead, cameras are often configured to send one frame every second or so to the server. This is especially true for cameras transmitting over a cellular network whose bandwidth may vary over time making the throughput unpredictable. In such cases, one can only analyze ultra low frame-rate videos out of which no motion features can be accurately extracted. Also, those cameras often have low resolution which makes localization and classification difficult.

To date, many object localization algorithms have been developed, and even more articles have been written on the topic. With the rise of deep learning methods (mostly convolutional neural nets), an exponential number of publications have been devoted to deep architectures implementing object recognition and localization methods [32,

31, 87, 40]. As a result, error rates on very challenging datasets, such as Pascal VOC [27], ImageNet [91] and Microsoft COCO [61], have decreased at a steady pace. Among other things, what makes deep learning so successful is the availability of large and well-annotated datasets. Unfortunately, despite the large number of publications devoted to traffic analytics, no such traffic dataset has been released to date. The lack of such a dataset has a number of implications, the most important one being that deep architectures trained on non-traffic oriented datasets such as ImageNet and COCO do not generalize well to traffic images.

Recognizing the importance of labeled datasets for the development of traffic analysis methods, we introduce the MIOvision Traffic Camera Dataset (MIO-TCD). It contains a total of 786,702 images : 137,743 high-resolution video frames with multiple vehicles in each frame and 648,959 vehicles cropped out of full frames. These images have been captured by hundreds of cameras deployed in urban and rural areas taking pictures at different periods of the year, at different times of the day, with different camera orientations and with various traffic densities. Many people have participated in the process of hand-annotating each image to provide a bounding box around each distinguishable object.

Leveraging this dataset, we demonstrate how well-trained, state-of-the-art deep learning methods can be used to localize and identify moving objects with high precision without having to rely on motion features. With classification accuracies of 96% and localization mean-average precision of 77%, we show that one can localize and recognize vehicles regardless of their orientation, scale, color and environmental conditions.

We believe that this work will have a substantial impact as there exists an urgent need for traffic monitoring systems working on still 2D images. We hope the MIO-TCD datset will have similar impact on the video surveillance community as the Caltech [24] and INRIA [21] datasets had on the pedestrian detection community, or ImageNet and COCO datasets – on the deep learning community.

The MIO-TCD dataset was the foundation of the Traffic Surveillance Workshop and Challenge held in conjunction with CVPR 2017. There were three primary goals to this challenge :

1. Provide the scientific community with a rich dataset to compare and test new

methods (the dataset will be regularly revised and expanded in the future and will maintain a ranking of methods).

2. Identify and rank the best traffic-oriented object localization and classification algorithms to date in various real-life conditions.
3. Identify remaining critical challenges in order to provide focus for future research.

In what follows, we elaborate on these goals, describe the dataset and the challenge results, and discuss unsolved challenges while offering avenues for future work.

A.3 Previous datasets

Today, we are witnesses to an exponential growth in the number of surveillance cameras are deployed along the roads of almost every country in the world. However, the images acquired by those cameras are the sole property of traffic management departments and are rarely released to the public. Consequently, few datasets have been made public for traffic analysis research. The most widely used datasets can be roughly divided into 3 categories, namely : (1) datasets of images taken by on-board cameras and mainly aimed at autonomous driving, (2) datasets of vehicle images taken by non-surveillance cameras and mainly oriented towards automatic recognition of high-resolution images from the Internet and (3) datasets of vehicle images taken by surveillance cameras. Here, we present a brief survey of existing vehicle detection and localization datasets.

On-board camera datasets

KITTI benchmark Dataset [30] [1] : This is a large dataset collected by an autonomous driving platform which addresses several real-world challenges, including : stereo vision calculation, optical flow estimation, visual odometry/SLAM, 3D object detection and 3D object tracking. This dataset consists of video frames captured by cars traveling both in rural areas and on highways.

1. http ://www.cvlibs.net/datasets/kitti/index.php

Cityscapes Dataset [17] [2] : This dataset focuses on the semantic segmentation of urban street scenes. It comes with 5,000 fully-annotated images and 20,000 weekly-annotated images. The annotated objects span 30 classes including eight different types of vehicles. Unfortunately, this dataset is limited to images acquired in urban areas (50 cities) and during summer daytime.

Tsinghua-Tencent Traffic-Sign Dataset [114] [3] : This is a large traffic-sign dataset containing 100,000 images with 30,000 traffic-sign instances. Images in this dataset cover various illumination and weather conditions but do not contain any labeled vehicles.

Vehicles captured by non-surveillance cameras

Stanford Car Dataset [52] [4] : This dataset contains 16,185 high-resolution images of 196 classes of cars. The vehicle classes include the brand, the model, and the year (e.g. 2012 Tesla Model S or 2012 BMW M3 coupe). This dataset is divided into 8,144 training images and 8,041 testing images. In addition to the large variety of vehicles, all pictures have excellent resolution and have been captured in good lighting conditions. However, none of the pictures were taken in a top-down orientation which is usually the case with surveillance cameras.

Comprehensive Cars Dataset (web) [109] [5] : This is one of the largest car dataset currently available. It comes with a total of 136,727 images showing entire cars and 27,618 images showing car parts. The dataset comprises 1,716 vehicle models as well as five different attributes (maximum speed, displacement, number of doors, number of seats, and type of car). Unfortunately, this dataset has the same limitations as the Stanford Car Dataset as its images were taken in a context far different than that of a surveillance camera (arbitrary orientation and filming 24/7).

2. https ://www.cityscapes-dataset.com/
3. http ://cg.cs.tsinghua.edu.cn/traffic-sign/
4. http ://ai.stanford.edu/ jkrause/cars/car_dataset.html
5. http ://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html

Traffic datasets from surveillance cameras

Comprehensive Cars Dataset (surveillance) [109] : This dataset contains 44,481 images of cars captured by frontal-view surveillance cameras. The ground truth provides the model and the color of each vehicle. The main limitation of this dataset comes from the frontal view of the pictures which makes it hard to generalize to an arbitrary camera orientation. Furthermore, this dataset focuses on cars, mini vans and pickup trucks, and does not contain any large articulated trucks, buses, motorcycles and pedestrians.

BIT-Vehicle Dataset [25] [6] : This dataset consists of 9,850 vehicle images. This is one of the most realistic datasets with images coming from real surveillance cameras. Vehicles are divided into six categories, namely bus, micro-bus, minivan, sedan, SUV and truck. Unfortunately, those images were all taken in a top-frontal view during daytime and clear weather, thus limiting the diversity needed for general traffic analysis applications.

Traffic and Congestions (TRANCOS) Dataset [36] [7] : This dataset is used to count the number of vehicles on highly-congested highways. It consists of 1,244 images with 46,796 annotated vehicles, most of which are partially occluded. Images were captured by publicly available video surveillance cameras of the Dirección General de Tráfico of Spain. Unfortunately, no vehicle type is provided.

GRAM Road-Traffic Monitoring (GRAM-RTM) Dataset [35] [8] : This is a benchmark dataset for multi-vehicle tracking. It consists of video sequences recorded by surveillance cameras under different conditions and with different platforms. Every vehicle has been manually annotated into different categories (car, truck, van, and big truck). Each video contains around 240 different objects.

Clearly, several publicly-available vehicle datasets contain images that do not come from surveillance cameras. The KITTI benchmark dataset, the Cityscapes Dataset and the Tsinghua-Tencent Traffic-Sign Dataset contain images captured by on-board cameras that can hardly be used to train traffic surveillance methods. The Stanford Car Dataset and the Comprehensive Car Dataset (web) contain high-resolution pic-

6. http ://iitlab.bit.edu.cn/mcislab/vehicledb/
7. http ://agamenon.tsc.uah.es/Personales/rlopez/data/trancos/
8. http ://agamenon.tsc.uah.es/Personales/rlopez/data/rtm/

figure A.1 – Sample images from the 11 categories of the classification dataset.

tures of vehicles mainly taken in frontal and side view and rarely in top-down view as is usually the case with traffic surveillance applications. Images from these datasets are usually applied to fine-grained vehicle analysis (counting the number of doors, identifying the vehicle brand and year, etc.). As for the Comprehensive Cars Dataset (surveillance), although it is one of the largest publicly-available surveillance dataset, its images come from a well-aligned frontal-view camera and show only one vehicle per image. It also shows images in daylight and good weather. Also, none of the datasets contains more than a couple of thousand images.

As for the BIT-Vehicle Dataset, it has up to two vehicles per frame but contains less than 10,000 images. The TRANCOS Dataset contains traffic jam images in which vehicles are too small to be categorized while the GRAM-RTM Dataset has only three video sequences.

A.4 MIO-TCD : Our Proposed Dataset

A.4.1 Dataset Overview

The MIO-TCD dataset contains a total of 786,702 images, 137,743 being high-resolution video frames showing multiple vehicles and 648,959 lower-resolution images of cropped vehicles. These images were acquired at different times of the day and different periods of the year by hundreds of traffic cameras deployed across Canada and the United States. Those images have been selected to cover a wide range of challenges and are typical of visual data captured in urban and rural traffic scenarios.

A.4. MIO-TCD : OUR PROPOSED DATASET

The dataset includes images : (1) taken at different times of the day, (2) with various levels of traffic density and vehicle occlusion, (3) showing small moving objects due to low resolution and/or perspective, (4) showing vehicles with different orientations, (5) taken under challenging weather conditions, and (6) exhibiting strong compression artifacts. This dataset has been carefully annotated by a team of nearly 200 people to enable a quantitative comparison and ranking of various algorithms.

The MIO-TCD dataset aims to provide a rigorous facility for measuring how far state-of-the-art deep learning methods can go at classifying and localizing vehicles recorded by traffic cameras. The dataset consists of two components : the classification dataset and the localization dataset. The classification dataset is used to train and test classification algorithms whose goal is to predict the kind of vehicle located in a low-resolution image patch. The localization dataset may be used to train and test algorithms whose goal is to localize and recognize vehicles located in the image.

MIO-TCD - Classification Dataset

The classification dataset contains 648,959 low-resolution images divided into 11 categories : Articulated Truck, Bicycle, Bus, Car, Motorcycle, Non-Motorized Vehicle, Pedestrian, Pickup Truck, Single-Unit Truck, Work Van and Background. As shown in Fig. A.1, each image contains one dominant object located in the middle of the image. The objects pictured in that dataset come in various sizes and were recorded in different parts of the year, different times of the day and from different viewing angles. The dataset was split into 80% training (519,164 images) and 20% testing (129,795 images). Note that the Car category contains vehicles of type sedan, SUV and family van.

The number of images in each category is listed in Table A.1. Since these images come from real footage, the number of samples per category is highly unbalanced as certain types of vehicles are more frequent than others. As such, almost half the images in the dataset fall into the car category, while the Bicycle, Motorcycle and Non-Motorized Vehicle categories contain roughly 2,000 training images and 500 testing images. We also randomly sampled 200,000 images to construct a background category.

figure A.2 – Sample images from the MIO-TCD localization dataset.

tableau A.1 – Size of each category in the classification dataset

Category	**Training**	**Testing**
Articulated Truck	10,346	2,587
Bicycle	2,284	571
Bus	10,316	2,579
Car	260,518	65,131
Motorcycle	1,982	495
Non-Motorized Vehicle	1,751	438
Pedestrian	6,262	1,565
Pickup Truck	50,906	12,727
Single-Unit Truck	5,120	1,280
Work Van	9,679	2,422
Background	160,000	40,000
Total	**519,164**	**129,795**

MIO-TCD - Localization Dataset

The localization dataset contains 137,743 images of which 110,000 are intended for training and 27,743 for testing. These images have various resolutions, ranging from 720×480 to 342×228. A total of 416,277 moving objects have been manually annotated with a bounding box and a category label. Except for Background, the same category labels as those in Table A.1 have been used. However, due to the fact that localizing and recognizing vehicles in a full video frame is more difficult than classifying images of already localized vehicles, we added a new Motorized Vehicle category which is unique to the localization dataset. This category contains all vehicles that are too small (or occluded) to be labeled into a specific category. Because of this category overlap, the classification dataset can be leveraged to improve the accuracy of localization methods.

Similarly to the classification dataset, images of the localization dataset came from hundreds of real traffic surveillance cameras. The category labels are thus unbalanced in similar proportions as those in Table A.1 (see Fig. A.2 for some examples).

A.4.2 Evaluation Metrics

Classification

Three metrics have been implemented to gauge performance of classification methods. The first metric is the overall accuracy (Acc) which is the proportion of correctly classified images in the whole dataset :

$$Acc = \frac{TP}{\text{total number of images}} \tag{A.1}$$

where TP is the total number of correctly-classified images regardless of their category. TP can also be seen as the trace of a confusion matrix such as the one in Fig. A.3.

Since the classification dataset is highly unbalanced, large categories such as Car and Background have an overwhelming influence on the calculation of the overall accuracy. We thus implemented three metrics which account for this imbalance, namely the mean recall (mRe), the mean precision (mPr) and the Cohen Kappa Score

($Kappa$) [16].

The mean recall and mean precision are obtained by averaging the recall and the precision of each category thus giving an equal weight to each category. This is done as follows :

$$mRe = \frac{\sum_{i=1}^{11} Re_i}{11} \qquad mPr = \frac{\sum_{i=1}^{11} Pr_i}{11}$$

where $Re_i = TP_i/(TP_i+FN_i)$ and $Pr_i = TP_i/(TP_i+FP_i)$ are the recall and precision for category i, and TP_i, FN_i, FP_i are the numbers of true positives, false negatives and false positives for the i-th category, respectively.

$Kappa$ is a measure that expresses the agreement between two annotators. In our case, the first annotator is a method under evaluation and the second annotator is the ground truth. The Cohen Kappa Score is defined as [16] :

$$Kappa = \frac{Acc - P_e}{1 - P_e} \tag{A.2}$$

where Acc is the accuracy (A.1) and P_e is the probability of agreement when both annotators assign random labels. $Kappa$ values are in the $[-1, 1]$ range where $Kappa = 1$ means that both annotators are in complete agreement, while $Kappa \leq 0$ means no agreement at all.

Localization

Following the Pascal VOC 2012 object detection evaluation protocol, we report localization results via precision/recall curves and use the average precision (AP) as the principal quantitative measure for each vehicle category. A detection is considered a true positive when the overlap ratio between the predicted bounding box and the ground-truth bounding box exceeds 50% ; otherwise, it is considered a false positive. We also report the mean-average precision (mAP) which is the mean of AP across all categories. We invite the reader to refer to the Pascal VOC development kit document [9] for more details on the AP metric.

The main difference between our localization challenge and other localization challenges is the occurrence of the Motorized Vehicle category. This category is used to

9. http ://host.robots.ox.ac.uk/pascal/VOC/voc2012/devkit_doc.pdf

label vehicles that are too small (or too occluded) to be correctly assigned a specific category. For example, a car seen from far away and whose size does not exceed a few pixels would be labeled as a Motorized Vehicle. In order not to penalize methods which would incorrectly label those small objects, every Motorized Vehicle labeled by one of following categories : Articulated Truck, Bus, Car, Pickup Truck, Single-Unit Truck and Work Van, is considered a true detection. In our implementation, we first identify every predicted bounding box whose overlap ratio with a Motorized Vehicle's ground-truth bounding box exceeds 50%, and re-assign its category label to Motorized Vehicle. Then, we compute the AP for each category.

A.5 Methods Tested

One of the overarching objectives of this paper is to identify how far state-of-the-art machine learning methods can go at classifying and localizing vehicles pictured by real traffic surveillance cameras. As mentioned earlier, this implies the analysis of low resolution 2D images with strong compression artifacts, recorded during daytime/-nighttime, in different seasons, under diverse weather conditions, and with various camera positions and orientations. As such, one can expect that some methods might be robust to those challenges while others may not. In order to identify solved and unsolved issues, we carefully benchmarked a series of state-of-the-art deep learning methods. In this section, we describe the methods that we tested on the MIO-TCD classification and localization datasets. Some methods have been published before the release of the MIO-TCD dataset while others have been designed specifically for this dataset's challenges [67].

A.5.1 Vehicle Classification

Pre-Trained CNN Features + SVM

The first series of methods aim at gauging how "different" is the statistical content of traffic images in the MIO-TCD dataset from other datasets, such as ImageNet. We did so by training a linear SVM classifier on CNN features obtained from ImageNet pre-trained CNN models. This is inspired by Razavian et al. [93] who demonstrated

that features obtained from deep CNN models could be the primary choice of a large variety of visual recognition tasks. We did so with the following six pre-trained deep models : AlexNet [54], InceptionV3 [100], ResNet-50 [40], VGG-19 [95], Xception [14] and DenseNet [44]. The layer we used to extract the features from is : ReLU of FC-7 in AlexNet and VGG-19, and the global pooling layer in InceptionV3, ResNet-50, Xception and DenseNet. Their corresponding feature dimensions are 4096, 4096, 2048, 2048, 2048, 1920. The python scikit-learn library [10] was used to train the linear SVM with $C = 1$.

Retrained CNN Models

Although features from the first layers of a CNN model are independent from the dataset it was trained on (mostly Gabor-like filters [110]), we retrained end-to-end all six CNN models on our classification dataset. Those models were all initialized with ImageNet pre-trained weights and were trained with the loss function proposed in the corresponding original papers. We used the Adam [51] optimizer with a learning rate of 10^{-3} that we empirically found more effective than other optimizers such as RMSprop or SGD. Training was done for a maximum of 50 epochs with a validation-based early stopping criteria with a patience of 10 epochs to prevent over-fitting. The batch size was adjusted to each model so it could fit on our 12GB Titan X GPUs. Please note that we included a series of batch normalization layers [45] in AlexNet and VGG-19 to speed up training. All models but DenseNet were implemented with the Keras library [13]. DenseNet was implemented with PyTorch [82].

We also implemented the following four training configurations to further improve results on our dataset :

— The first configuration is a basic training with normal sampling of the data.

— The second configuration involves data augmentation using horizontal flipping and randomized shearing and zooming to enrich the training dataset.

— Since the dataset is highly unbalanced (see Table A.1), we used data augmentation with uniform sampling. That is, at each epoch we used an equal number of images from each class to prevent large classes, such as *Car*, *Pickup Truck*

10. scikit-learn.org/

and *Background* from gaining too much importance over smaller classes, such as *Motorcycle*.

— As suggested by Havaei et al. [39], we implemented a two-phase training procedure. In the first phase, we used data augmentation with uniform sampling. In the second phase, we froze the entire network except for the last layer which was retrained with data augmentation and normal sampling.

MIO-TCD Classification (Ensemble Models)

In the wake of the 2017 CVPR MIO-TCD Challenge [67], several methods have been designed for the sole purpose of classifying traffic images. Interestingly, all of those methods involve a combination of several deep learning models. Kim and Lim [50] proposed a bagging system where several CNN models are trained on a random subset of the MIO-TCD dataset. Their final result is obtained with a weighted majority vote to compensate for the unbalanced nature of the dataset. Lee and Chung [58] proposed an ensemble method which combines 3 convolutional nets (AlexNet, GoogleNet and ResNet18) trained on 18 different sets of data. GoogleNet was trained on 12 subsets of the dataset (aka the local nets) and AlexNet, GoogleNet and ResNet18 were trained on the entire dataset but with different image sizes (aka the global nets). At the test time, the networks are selected with a gating function and combined with a softmax layer. Jung et al. [48] proposed an ensemble model according to which several deep residual networks are jointly trained. The main novelty of their method lies within its loss function which allows to train every ResNet simultaneously. Theagarajan et al. [101] also proposed an ensemble of ResNet models. The authors implemented a weighted loss function to account for the unbalanced nature of the dataset. They also implemented a patch-wise logical reasoning process to disambiguate classes that are close to each other like trucks and buses.

A.5.2 Vehicle Localization

Recent CNN-Based Methods

Recently, CNN-based localization methods established state-of-the-art performance on several object localization datasets. In this paper, we evaluated Faster R-CNN [87],

SSD-300, SSD-512 [63], YOLO [84] and YOLO-v2 [86].

Faster R-CNN is an improved version of Girshich et al.'s R-CNN [32] and Fast R-CNN [31] methods. Unlike R-CNN and Fast R-CNN, that use a selective search [102] to generate object bounding box proposals, Faster R-CNN has a region proposal network that can directly estimate proposals based on the CNN feature maps thus making it end-to-end trainable. Liu *et al.* [63] improved upon the Faster R-CNN model with their SSD (Single Shot MultiBox Detector) framework which generates object proposals with feature maps from multiple layers. As for YOLO (You Only Look Once) [84], it takes a 448×448 image as input and outputs object detections within a 7×7 grid. Later on, Redmon *et al.* [86] integrated anchor boxes (for computing potential bounding boxes) into their YOLO model. We refer to this model as YOLO-v2.

Similarly to the comparison of pre-trained CNN features and retrained CNN models for classification, we trained all localization models with and without updating the weights inherited from an ImageNet pre-trained model to observe the efficiency of ImageNet features on vehicle localization (i.e. we only trained the layers that weren't initialized with the ImageNet weights). We trained Faster R-CNN end-to-end for 300,000 iterations with code provided by the authors [11]. As recommended by the original paper, we used VGG-16 as a pre-trained model. Similarly, for SSD-300 and SSD-512 (i.e., SSD with input images resized to 300×300 and 512×512, respectively) we used the code released by the authors [12]. The SSD-300 model was trained for 120,000 iterations with a batch-size of 32, and the SSD-512 model was trained for 240,000 iterations with a batch-size of 16. YOLO was trained using the Darknet deep learning toolbox [13]. Both YOLO and YOLO-v2 were trained for 80,000 iterations with a batch-size of 64. As YOLO-v2 needs pre-clustered anchor bounding boxes, we evaluated two kinds of anchor boxes : boxes computed on the Pascal VOC datasets (referred to as YOLO-v2(P)) and on our localization dataset (referred to as YOLO-v2(M)).

11. https ://github.com/rbgirshick/py-faster-rcnn
12. https ://github.com/weiliu89/caffe/tree/ssd
13. https ://pjreddie.com/darknet/

MIO-TCD Localization

We report results from two localization methods designed specifically for the MIO-TCD dataset. The first one is from Wang et al. [107] which improves methods such as Faster R-CNN and SSD by leveraging the scene context. The idea is to combine the softmax score of these methods with a context term which the authors model with a k-NN algorithm. The second method is from Jung et al. [48] which combines the results computed from multiple R-FCN models [20] trained with different backbones (ResNet-50 and ResNet-101) together with a joint non-maximal suppression step to localize vehicles.

A.6 Experimental Results

In this section, we report experimental results obtained on the MIO-TCD classification and localization datasets with the deep learning methods presented earlier.

A.6.1 Vehicle Classification

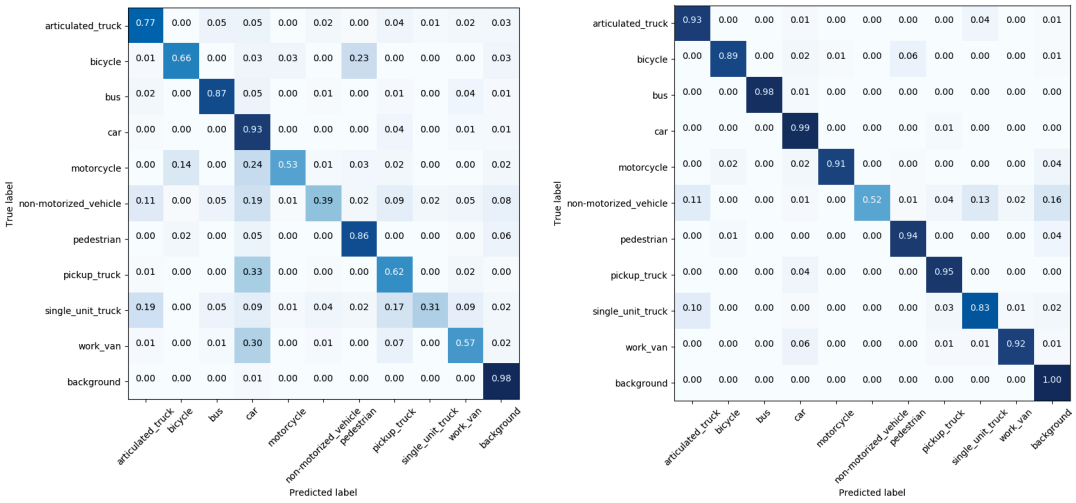


figure A.3 – Confusion matrices obtained on the classification dataset with pre-trained ResNet-50 features + SVM on left and the ensemble model by Jung *et al* [48] on right.

tableau A.2 – Evaluation metrics for six pre-trained models used with linear SVM classifiers on the classification dataset.

	Acc	*mRe*	*mPr*	*Kappa*
AlexNet	0.82	0.49	0.55	0.72
Inception-V3	0.84	0.57	0.64	0.75
ResNet-50	**0.89**	**0.69**	0.74	**0.83**
VGG19	0.83	0.66	0.59	0.75
Xception	0.87	0.54	0.76	0.78
DenseNet	0.86	0.51	**0.82**	0.78

Pre-Trained CNN Features + SVM

Results obtained with SVM trained on features extracted from six pre-trained CNN models are shown in Table A.2. All six models have an accuracy of more than 80%, which is surprisingly high considering the different nature of ImageNet and MIO-TCD datasets. However, careful analysis reveals that these methods have relatively low mean recall, mean precision and Kappa score. To understand this situation, one has to consider the confusion matrix of ResNet-50 shown in Fig. A.3. While the *Car* and *Background* categories have an accuracy of more than 90%, others, such as *Non-Motorized Vehicle*, *Motorcycle* and *Single-Unit Truck* categories, suffer from very low accuracy.

These numbers can be explained by the unbalanced nature of the dataset, as the *Car* and *Background* categories contain more than 80% of all images. In this case, large classes strongly influence the decision boundaries to the detriment of smaller classes. This also explains why several categories have a large proportion of samples wrongly classified as *Car*. It is the case for *Work Van* for which 30% of its images are classified as *Car*. A detailed analysis revealed that features trained on ImageNet do not discriminate family vans (labeled as *Car* in the dataset) from *Work Vans*. Confusion also happens between *Bicycle* and *Pedestrian*, as well as within the group of *Articulated Truck*, *Pickup Truck* and *Single-Unit Truck*.

Based on these results, we conclude that SVM trained on ImageNet CNN features is accurate at classifying large classes but this does not generalize well to smaller classes.

Retrained CNN Models

tableau A.3 – Evaluation metrics for retrained CNN models on the classification dataset in four different configurations. 'N' stands for normal sampling, 'U' for uniform sampling, 'D' is for using data augmentation and 'T' is a two-phase training procedure.

	Acc				*mRe*				*mPr*				*Kappa*			
	N	D+N	D+U	D+U(T)	N	D+N	D+U	D+U(T)	N	D+N	D+U	D+U(T)	N	D+N	D+U	D+U(T)
AlexNet	0.869	0.892	0.820	0.883	0.631	0.560	0.826	0.417	0.546	0.692	0.631	0.899	0.796	0.832	0.743	0.807
Inception-V3	0.947	0.962	0.929	0.959	0.809	0.828	0.888	0.872	0.792	0.848	0.763	0.889	0.918	0.941	0.892	0.937
ResNet-50	0.938	0.967	**0.959**	**0.969**	0.795	0.871	0.892	**0.877**	0.734	0.858	0.865	0.903	0.902	0.948	**0.936**	**0.952**
VGG-19	0.941	0.956	0.887	0.940	0.773	0.824	0.654	0.817	0.794	0.853	0.838	0.841	0.909	0.932	0.831	0.907
Xception	0.958	**0.976**	0.941	0.961	0.846	**0.908**	0.822	0.819	0.794	0.906	**0.900**	0.836	0.935	**0.963**	0.910	0.939
DenseNet	**0.970**	0.973	0.952	0.954	**0.889**	0.870	**0.896**	0.825	**0.886**	**0.916**	0.850	**0.908**	**0.953**	0.958	0.927	0.929

Table A.3 reports evaluation metrics for the six models trained in four different training configurations. As can been seen, there is a substantial performance increase in comparison to the SVM results from Table A.2. Results show that data augmentation improves the Kappa score of every model, especially ResNet-50. Note that although data augmentation may reduce the mean recall of a certain method, this is compensated by a larger increase of the mean precision. However, the use of uniform sampling (+U) with and without the two-phase training procedure (T) does not improve results over the normal sampling (+N).

A careful inspection reveals that uniform sampling has a positive impact on small categories (such as Motorcycle, for example), but decreases the performance for large categories.

Overall, Xception and DenseNet with data augmentation and normal sampling are the best methods with very similar performance. VGG-19, ResNet-50 and Inception-V3 also get very accurate results with Kappa scores above 0.93.

MIO-TCD Classification (Ensemble Models)

Table A.4 reports results submitted to the 2017 CVPR MIO-TCD Challenge. These are ensemble methods which combine the outputs of different models. As can be seen, all these methods have similar performance with accuracy of about 0.98 and Kappa score of almost 0.97. With these results being only marginally better than those obtained with Xception and DenseNet, we conclude that the combination of several models does not bring much for a dataset such as MIO-TCD.

tableau A.4 – Evaluation metrics for ensemble models from the 2017 MIO-TCD Classification challenge.

	Acc	*mRe*	*mPr*	*Kappa*
Kim and Lim [50]	0.9786	0.9041	0.9355	0.9666
Lee and chung [58]	0.9792	0.9024	0.9298	0.9675
Jung *et al* [48]	**0.9795**	0.8970	**0.9530**	**0.9681**
Theagarajan *et al* [101]	0.9780	**0.9190**	0.9439	0.9658

Error Analysis

Results from Tables A.3 and A.4 reveal that despite large illumination variations between images, compression artifacts, arbitrary vehicle orientation, poor resolution and inter-class similarities, the classification of traffic vehicles seems almost solved. However, in-depth analysis of the top-performing methods reveals some unsolved issues. In Fig. A.3, we show the confusion matrix for the method by Jung *et al.* [48] whose performance is globally similar to that obtained by other top performing methods. As one can see, *Non-Motorized Vehicles* are poorly handled. Images of *Non-Motorized Vehicles* in our dataset include a wide variety of trailers pulled by a vehicle, typically a car or a pickup truck. As shown in Fig. A.4 (second row, third column), *Non-Motorized Vehicles* are often wrongly classified as a single-unit or an articulated truck, as their shapes are very similar.

Without much surprise, methods also get confused between categories with similar visual characteristics, such as the Work Van class and the Car class (more specifically, family vans considered to belong to the Car class) or the Articulated Truck and the Single-Unit Truck. They also get confused with vehicles that look unusual. For example, in Fig. A.4, the blue car with a black top (second row, first column) gets wrongly classified as a pickup truck and the pickup truck with a cap (third row, fist column) is wrongly classified as a car. Also, classes with small objects such as *Pedestrian*, *Bicycle* and *Motorcycle* often suffer from heavy compression artifacts and are thus more likely to be mis-classified.

figure A.4 – Examples of failure cases from top-performing methods for every class where the yellow label (top) is the ground truth and the white label (bottom) is the predicted class.

tableau A.5 – Average precision (AP) of localization for Faster R-CNN, SSD, YOLO and two methods submitted to the MIO-TCD Challenge on localization. ("w/o" : without updating the weights inherited from an ImageNet pre-trained model)

	mAP	Articulated Truck	Bicycle	Bus	Car	Motorcycle	Motorized Vehicle	Non-Motorized Vehicle	Pedestrian	Pickup Truck	Single-Unit Truck	Work Van
Faster R-CNN (w/o)	6.3	4.5	2.0	3.0	5.5	3.7	6.6	0.9	0.4	10.0	2.4	2.6
SSD-300 (w/o)	36.4	44.1	52.2	68.0	55.0	38.5	29.4	3.2	10.7	55.6	19.3	24.7
SSD-512 (w/o)	34.8	45.2	25.6	71.0	61.9	30.0	37.0	0.8	12.2	61.0	15.8	22.0
YOLO-v1 (w/o)	31.3	48.2	19.2	78.5	50.6	15.4	17.3	6.3	2.3	61.8	12.7	31.9
YOLO-v2(P) (w/o)	46.3	54.6	53.5	82.8	61.4	56.8	26.8	20.5	9.9	68.8	30.2	43.5
YOLO-v2(M) (w/o)	47.7	60.9	54.2	85.9	62.0	60.7	27.1	19.2	8.8	69.9	32.3	43.7
Faster R-CNN	70.0	85.9	78.4	95.2	82.6	81.1	52.8	37.3	31.3	89.0	62.5	73.6
SSD-300	74.0	90.6	78.3	95.7	91.5	78.9	51.4	55.2	37.3	90.7	69.0	75.0
SSD-512	**77.3**	**92.1**	**78.6**	**96.8**	**94.0**	**82.3**	**56.8**	**58.8**	**43.6**	**93.1**	**74.0**	**80.4**
YOLO-v1	62.7	82.7	70.0	91.6	77.2	71.4	44.4	20.7	18.1	85.6	58.3	69.3
YOLO-v2(P)	71.5	86.7	78.4	95.2	80.5	80.9	52.0	56.5	25.7	84.6	70.0	75.7
YOLO-v2(M)	71.8	88.3	**78.6**	95.1	81.4	81.4	51.7	56.6	25.0	86.5	69.2	76.4
Wang *et al.*[107]	77.2	91.6	79.9	96.8	**93.8**	83.6	56.4	58.2	42.6	**92.8**	73.8	79.6
Jung *et al.*[48]	**79.2**	**92.5**	**87.3**	**97.5**	89.7	**88.2**	**62.3**	**59.1**	**48.6**	92.3	**74.4**	**79.9**

A.6.2 Vehicle Localization

Recent CNN-Based Methods

The average precision of localization for Faster R-CNN, SSD-300, SSD-512, YOLO-v1, YOLO-v2(P) and YOLO-v2(M) methods is shown in Table A.5.

Methods followed by "(w/o)" were trained without updating the weights inherited from a pre-trained ImageNet model. Note that when training Faster R-CNN with frozen layers, we found that it failed to converge. This behavior (also reported in [94]) is mainly due to gradient issues when using RoI pooling layers. For the other models, we see that training them in full is highly beneficial, with a very significant boost of mAP (24.1–42.5% improvement).

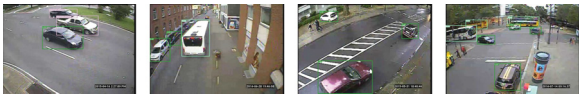
From this point, we only discuss results obtained with fully-trained models. As can be seen, the SSD methods outperform both Faster R-CNN and YOLO, with SSD-512 being the best-performing method with a mAP of 77.3%.

Results for SSD-300 and SSD-512 show that increasing input image resolution from 300×300 to 512×512 improves the mAP by 4%. Also, YOLO-v2 has the mAP 10% higher than YOLO-v1 thus showing that anchor boxes are useful features. Furthermore, results for YOLO-v2(P) and YOLO-v2(M) show that anchor boxes computed on our localization dataset marginally improve mAP over anchor boxes pre-computed from the Pascal VOC dataset.

Here again, the largest classes, namely *Car*, *Pickup Truck*, *Articulated Truck* and *Bus* get the best results with an average precision above 80% for almost every method, while *Motorized Vehicle*, *Non-Motorized Vehicle* and *Pedestrian* are the three categories with the lowest average precision. The main challenge with *Motorized Vehicle* and *Pedestrian* classes stems from the small size of vehicles that are likely to be confused with the *Bicycle* and *Motorcycle* categories. As for the *Non-Motorized Vehicle*, similarly to the classification dataset, it is often confused with the *Articulated Truck* and *Single-Unit Truck* classes.

In Fig. A.5, we show some detection results for different methods. While most methods can accurately localize large and well-contrasted vehicles, we can see that Faster R-CNN is prone to false detections while YOLO-v1 and YOLO-v2 suffer from mis-detections of small objects (typically, pedestrians).

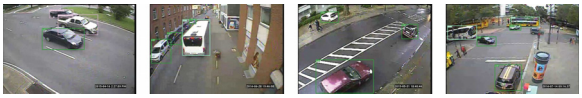
(a) Faster R-CNN



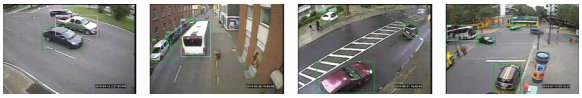
(b) SSD-300

(c) SSD-512

(d) YOLO-v1



(e) YOLO-v2 (Pascal VOC)



(f) YOLO-v2 (MIO-TCD)

figure A.5 – Detection examples on the localization dataset for Faster R-CNN, SSD-300, SSD-512, YOLO, YOLO-v2 (Pascal VOC) and YOLO-v2 (MIO-TCD). We only show detections with probability scores higher than 0.6.

MIO-TCD Localization

At the bottom of Table A.5, are shown localization metrics for methods by Wang *et al.* [107] and Jung *et al.* [48] submitted to the MIO-TCD Challenge. Both methods attain excellent performance, with the Jung *et al.* method achieving the best mAP of 79.2% and outperforming state-of-the-art CNNs for almost all vehicle classes.

Detailed Analysis

We now thoroughly analyze the influence of object scale on the performance of localization methods as well as the nature of false detections. We do so via the Microsoft COCO's evaluation procedure [61] and the object detectors' protocol by Hoiem et al. [42].

Scale Every object has been classified as belonging to one of 3 scales : small objects with bounding box area below 32^2, medium objects with the area between 32^2 and 96^2, and large objects with the area larger than 96^2. The average precision for each of these scales is reported in Table. A.6. As can be seen, all methods in the table are ill-suited for detecting small objects, in our case *Pedestrian*, *Bicycle*, *Motorcycle* classes as well as vehicles seen from a distance (see Fig. A.2 for examples of small vehicles due to perspective). Furthermore, when increasing the overlap ratio for correct detections from 0.5 to 0.75, we find that the AP of Faster R-CNN and YOLO decreases by almost 30%, while for SSD it decreases by around 15%. This means that the bounding boxes estimated by the SSD method are tighter around the ground-truth bounding boxes.

False positives To examine the nature of false positives, we follow the methodology of Hoiem et al. [42] according to which each prediction is either correct or wrongly classified into one of the following errors :

— **Localization** : the predicted bounding box has a correct label but is misaligned with the ground-truth bounding box ($0.1 < \text{Overlap} < 0.5$).

— **Similarity** : the predicted bounding box has Overlap > 0.1 with a ground-truth bounding box but its predicted label is incorrect. However, the predicted label belongs to one of three similarity classes (groups of similar classes), na-

tableau A.6 – Average precision of localization computed using Microsoft COCO's evaluation protocol.

	Average Precision				
	Overlap		Scale		
	0.5	0.75	small	medium	large
Faster R-CNN	70.0	38.5	14.3	40.2	55.1
SSD-300	74.3	57.1	21.5	53.3	69.0
SSD-512	**77.6**	**61.9**	**28.2**	**57.3**	**72.5**
YOLO-v1	62.6	34.0	11.3	32.4	52.7
YOLO-v2(P)	71.3	42.7	15.7	41.3	59.3
YOLO-v2(M)	71.8	43.0	16.0	41.7	61.3
Wang *et al.* [107]	77.4	**59.9**	**26.6**	**55.6**	60.7
Jung *et al.* [48]	**79.3**	58.8	26.5	54.9	**69.3**

mely : {*Articulated Truck, Pickup Truck, Single-Unit Truck*}, {*Bicycle, Motorcycle, Pedestrian*}, and {*Car, Work Van*}.

— **Other** : the predicted bounding box has a Overlap > 0.1 with a ground-truth bounding box but its predicted label is incorrect and does not fall within a group of similar classes.

— **Background** : all other false positives are classified as background, mainly confused with unlabeled objects.

Fig. A.6 shows the frequency of occurrence of each type of error for the SSD-300 method. For the similarity class {*Articulated Truck, Pickup Truck, Single-Unit Truck*}, the *Articulated Truck* and *Single-Unit Truck* classes are likely to be confused with each other, while the *Pickup Truck* is confused with other classes, mostly *Car*. As for the {*Car, Work Van*} similarity class, we find that 66% of *Work Van* false positives are wrongly classified as *Car* for the reason mentioned before (*Work Van* is often confused with a family van). As for *Car* false detections, the confusion is mostly with *Background*. For the {*Bicycle, Motorcycle, Pedestrian*} similarity class, the *Bicycle* and *Motorcycle* classes are often confused with each other, while *Pedestrian*, due to small scale, suffers from localization errors. As for the *Bus*, *Motorized Vehicle* and *Non-Motorized Vehicle* classes, they all suffer from confusion with other categories.

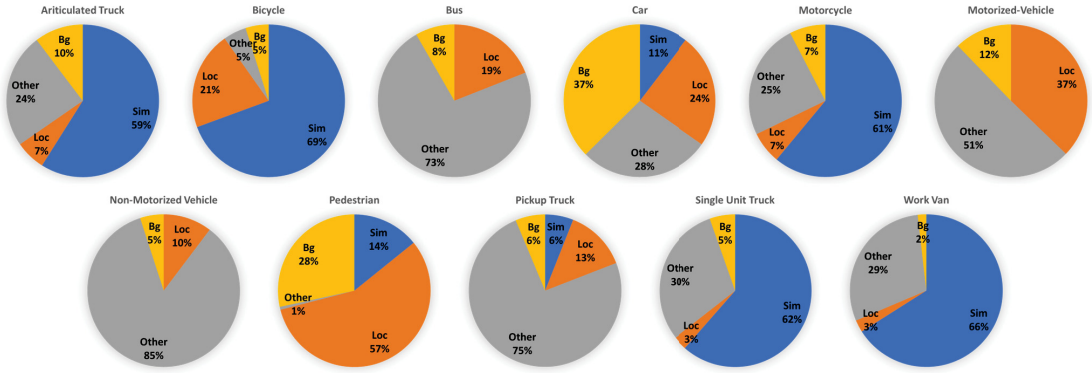


figure A.6 – Analysis of false detections by the SSD-300 method. Each pie-chart shows the fraction of top-ranked false positives of each category due to poor localization (Loc), confusion with similar categories (Sim), confusion with other categories (Other), or confusion with background or unlabeled objects (Bg).

A.7 Discussion and Conclusions

In this paper, we introduced the MIOvision Traffic Camera Dataset (MIO-TCD), the largest dataset to date for motorized traffic analysis. The dataset consists of two parts : a "localization dataset", containing full video frames with bounding boxes around traffic objects, and a "classification dataset", containing crops of 11 types of traffic objects.

We evaluated several state-of-the-art deep learning methods on the MIO-TCD dataset. Results show that well-trained models reach impressive accuracy and Kappa scores of more than 96% on the classification dataset and a mean-average precision of 79% on the localization dataset.

While Xception and DenseNet with data augmentation are the best classification methods, VGG-19, RestNet-50 and Inception-V3 attain very good results as well. As for the ensemble models, they reach, for all practical purposes, the same scores as Xception and DenseNet. A careful inspection of results reveals that Non-Motorized Vehicles is the only problematic class with a precision below 80%. Other errors in the top results can be explained by the confusion between classes with similar visual characteristics such as Single-Unit Truck and Articulated Truck.

As for localization methods, the method by Jung et al. [48] gets the best scores

(mAP=79.2%) but is closely followed by SSD-512 (mAP=77.3%). A detailed analysis reveals that errors often happen between similar classes or are due to a mis-alignment of the predicted bounding box (overlapping ratio below 0.5).

In light of these results, we may conclude that state-of-the-art deep learning methods exhibit a capacity to localize and recognize vehicles from single video frames without the need for dynamic features captured by video, as was required to date. This opens the door to new, low-frame-rate video analytics applications such as traffic statistics, traffic density estimation, car counting, and anomaly detection.

Although deep models achieve very promising performance, a number of challenges remain, among them : similarly-looking vehicles, unbalanced data, false detections and small vehicles. We conclude the paper by discussing these challenges below.

1. *Similarly-looking vehicles* : To differentiate vehicles with similar appearance, such as bicycles and motorcycles, it may be necessary to identify semantic features in addition to global features.
2. *Unbalanced data* : Experiments show that ensemble models can deal with this issue to some extent, at the cost of significantly more computations. It would be beneficial to develop algorithms that leverage the benefits of ensemble models at reduced computational load.
3. *False detections* : The appearance and position of different vehicles is highly correlated with scene layout, such as road direction. It would be preferable to embed the layout information into the localization model as a means of enhancing detection and reducing false positives.
4. *Small vehicles* : The localization results indicate that large vehicles are more easily detected than small vehicles. Adversarial training or metric learning methods projecting vehicles of various sizes into the same feature space could be an avenue towards improving the localization and classification accuracy of small vehicles.

Annexe B

Single-frame all-in-one model for traffic analysis

L'article suivant est une ébauche en vue d'une soumission au journal T-ITS sur une nouvelle application de l'apprentissage profond pour estimer l'orientation des véhicules photographiés par des caméras de surveillance. Il décrit le processus de création de la base de données, les modèles utilisés ainsi qu'une analyse approfondie des résultats.

Andrew Aachkar de la compagnie Miovision a fourni les données et participé à la rédaction. Pierre-Marc Jodoin a eu l'idée d'utiliser les *oriented distribution functions* comme *apriori* et a participé à la rédaction. Pour ma part, j'ai implémenté les modèles, effectué les tests et j'ai participé à la rédaction.

Nos contributions sont les suivantes :

— Une mise à jour de la base de données MioTCD pour y inclure des informations sur le mouvement.
— La présentation de deux nouveaux modèles inspirés de SSD[63].
— Une nouvelle composante en entrée pour donner un *apriori* au modèle.
— Une analyse approfondie sur la difficulté que représente l'estimation d'orientation et de mouvement.

Single-frame all-in-one model for traffic analysis

Frédéric Branchaud-Charron[1], Andrew Aachkar[2], Pierre-Marc Jodoin[1]

[1] *Département d'informatique, Université de Sherbrooke, Sherbrooke, Canada*

[2] *Miovision inc., Waterloo, Canada*

B.1 Abstract

Most traffic analysis systems rely on machine learning models to achieve high level tasks such as tracking or traffic estimation. In practice, due to the higher cost and complexity, these systems can't afford high frame-per-second (fps). In consequences methods that rely on video and motion features are unusable in this context. In this work, we propose an all-in-one model for vehicle detection from a single frame. Our model predicts localization, classification, orientation and whether the vehicle is idle or moving. Our method also benefits from an online learning component which creates a strong prior on the orientation over time. Our contributions are three-fold : an end-to-end model for vehicle detection, an online learning component for orientation estimation and an indepth analysis of our orientation prediction.

B.2 Introduction

Nowadays, most traffic monitoring systems rely on motion features to perform basic tasks such as counting the number of vehicles, estimating traffic speed or estimating the distribution of classes of vehicle. [108, 113, 62] This is a limitation for many large-scale camera networks that cannot afford the high framerate required by these methods thus making them unusable in this context.

Without motion features, it is not trivial to estimate the trajectory of a vehicle nor estimate whether the vehicle is idle or not. This work propose to estimate those, in addition to the standard localization and classification, from a single frame. Those tasks are useful to approximate the motion features that are lacking.

In addition, we introduce an online learning component to provide the model with an approximate context of the scene that it is asked to monitor. This online learning component is represented as an **orientation distribution function** (ODF). While

we do not have the motion information, we can build a strong prior from the orientation estimation map. This prior has two functions : it first gives an approximation of where we should look for objects in the scene and second, it approximates the motion information of the scene.

To summarize, our contributions are three-fold.

— We use an online learning component as a prior to help the network.

— We propose a network which estimates four metrics usable by traffic monitoring systems.

— We provide an in-depth analysis of the scene context and how it is useful to estimate the orientation or the status of a vehicle.

B.3 Previous works

B.3.1 Localization

Following the popularity of deep learning, many models were published on the task of localization. Among them, Single-Shot Detector (SSD) [63] is one of the most commonly used because of its performance and it's fast inference time. SSD is built upon VGG-16citeto extract visual features then, a combination of low-level and high-level features are used to predict the box coordinates and its associated class. The box coordinates are determined by selecting an anchor which represents the aspect ratio of the box. The network then predicts a translation and a scaling to augment the selected anchor. Finally, the predicted boxes are fed to a non-max suppression algorithm to remove overlapping boxes. With an input size of 300×300 it achieves 74.3% mean average precision (mAP) on Pascal VOC 2007citewith an inference time of 0.017 seconds.

B.3.2 Orientation estimation

To our knowledge, there has been no previous work done on orientation estimation on full frames. Current techniquesciteprocess ground-truth crops of the object before estimating the orientation. Hara *et al.*[38] is one of the first work to use CNNs to perform this task. They designed a network that outputs a vector $(\cos\theta, \sin\theta)$ and

then tried multiple losses such as the L1 norm and $1-\cos\theta$. Finally they experimented with a model that learns from a finite number of discrete orientations, they train multiple outputs with different starting orientation angles. The predictions of those outputs are combined using a mean-shift method.

B.3.3 Idling detection

To the best of our knowledge, there has been no previous work on estimating the status (idle or moving) of a car from a single image. [5] propose a method which takes multiple frames from an infrared camera to estimate whether a car is parked or idling. To do so, they first track vehicules across all frames. For each car, they crop the region around the car, before using a ConvRNN which outputs the decision based on the infrared crops. Our model is different since it only differentiate between moving and stationnary vehicules. We are also working on a single-frame setting which makes this method unusable.

B.3.4 Orientation Distribution Function

Orientation distribution functions (ODFs) are a probabilistic function which model the probability of a specified orientation. In the discrete case, we discretize the circle space in b bins where each bin represents the probability that the orientation is within the range associated with.

B.4 Model

Our model as shown in Fig. B.1 is an extended version of the popular Single-Shot Detector (SSD)[63]. In addition to the usual outputs, we add a new branch which perform a regression on the orientation and a binary classification on the idle state. Those two outputs are computed at each scale similar to the localization and classification heads. Those two new heads add 275k parameters to the original SSD which does not impact the inference time of the model significantly. The regression outputs a single scalar between 0 and 1 and is learned through our cosine loss (Eq. B.1).

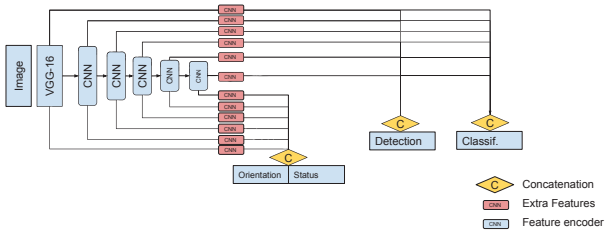


figure B.1 – SSD architecture.

$$L_c(y, \hat{y}) = 1 - \arccos\left(v(y) \cdot v(\hat{y})\right) \tag{B.1}$$

Where $v(y) = (\cos(2\pi y), \sin(2\pi y))$. We weights the loss by the idle component. The groundtruth is not reliable for idle cars thus their importance is lowered.

$$L_o(y_o, \hat{y_o}, y_i) = \lambda y_i * L_c(y_o, \hat{y_o}) + (1 - \lambda)(1 - y_i) * L_c(y_o, \hat{y_o}) \tag{B.2}$$

Where $\lambda = 1/5$ in our experiences.

The idle component is learned throught crossentropy :

$$L_i(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - y)). \tag{B.3}$$

The localisation and classification losses are not modified. Our total loss is :

$$L = L_{loc} + L_{cls} + L_o + L_i. \tag{B.4}$$

B.4.1 Orientation estimation with ODF

We can take into account the stationnary state of the camera to create a strong prior on the orientation. We model this prior using an orientation distribution function (ODF). It is describe as a grid where the orientation space is discretize in b bins. This new component is added as a new input to our model.

The ODF is inserted after the VGG-16 backbone and is progressively merged at each scale with each features map as shown in Fig. B.2. This adds few parameters (51k) which still allows the model to be used in real-time.

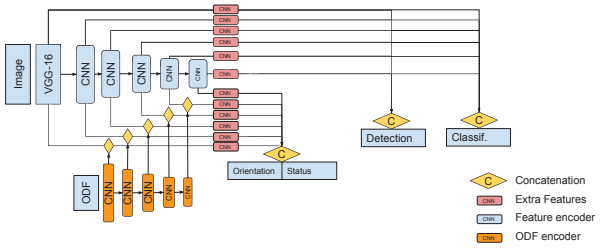


figure B.2 – SSD architecture with ODF.

B.5 Experimentation

B.5.1 Experimental setup

Dataset creation

MioTCD[66] is localization dataset used in a challenge at the international conference CVPR 2017. It contains more than 130k fully-annotated frames of urban scenes with 11 classes such as car, pedestrian or work van. The scenes propose a high variety of image quality, weather and context. State-of-the-art methods showed high performance on this dataset achieving 77% mAP when the challenge concluded.

From the video sequences used to generate this dataset, we computed the optical flowcitenear the annotations, giving us an estimation of the orientation around moving vehicles. To compute this estimation, we use the most common angle where the magnitude of the vector is positive. This also gives us an estimation of whether the car is idle or not, since no optical flow can be retrieved at those positions.

To get an estimation of the orientation of idle vehicles, we trained an Xception[14] model pretrained on Imagenet to classify each crop and predict their orientation. We used the same loss as in Eq. B.1. This models achieves a 0.308 mean radians error on our test set. We then use this model to perform inference on idle cars. We also tried the MeanShift method with $N = 8, M = 9$ and we obtain a 0.4848 mean radians error. Orientation on stationary vehicles is important during training if we also want to estimate orientation of stationary vehicles which we do.

tableau B.1 – Inference time for each method.

Model	Slow factor
SSD	1x
SSD-OE	1.09x
SSD-ODF	1.10x

Other methods

Since there is no directly available models to compare our method with, we used two-steps approaches. We first extract boxes from a standard SSD-300 trained on MioTCD. It achieves a 75.2% mAP on the test set. Then, we used the Xception[14] model and the MeanShiftcitemethod to compute the orientation on the predicted boxes.

Model training

All localization models were trained for 500k iterations and we used $b = 20$ for the ODF model. The models are trained using the SGD optimizer with a learning rate of 0.001, a momentum of $5e{-}4$ and a batch size of 32. We add a L2 regularizer onto the weights with a factor of $5e - 4$. During training, we lowered the learning rate by a factor of 0.1 at iteration 280000, 360000 and 400000. We then generate boxes and keep all boxes where the confidence is higher than 0.01. We process the boxes with a non-maximum suppression algorithm and merge boxes where the $IoU < 0.45$. SSD models use pretrained weights from Pascal VOC 2007cite.

Inference time

Our model adds 325k parameters to the original SSD model, but as we see in Table B.1, the model using ODF is only 10% slower than the original model. Most parameters comes from the added output.

B.5.2 Comparison

We first compare the localization score between all models. It is common that adding new outputs decrease the performance of another output. As can be seen in

tableau B.2 – mAP achieved by each model.

Method	mAP(%)
SSD-300	75.2
SSD-300-OE	74.3
SSD-300-OE w/ ODF	74.2

Table B.2, the added component does not have a meaningful impact on the mean Average Precision (mAP).

Methodology

We now present our method to evaluate all models. We first match every prediction with a groundtruth annotation if possible. A prediction can be matched if it has an $IoU > 0.5$ with another boxes. We selected the box with the highest IoU. Predictions without matching annotation are simply dropped. We then compute multiple metrics such as the arccosine distance Eq. B.5, the cosine distance Eq. B.1, the dot product and the number of prediction off by more than 0.3 radiansEq. B.6. Since our groundtruth for idle vehicle is not reliable, we do not use those annotations for evaluation :

$$f(X, Y) = \mathbb{E}_{x \in X, y \in Y} \arccos(x \cdot y), \tag{B.5}$$

$$f(X, Y) = \mathbb{E}_{x \in X, y \in Y} \delta(\arccos(x \cdot y)), \tag{B.6}$$

$$\delta(x) = \begin{cases} 0, & x \leq 0.3 \\ 1, & \text{otherwise} \end{cases} \tag{B.7}$$

where X, Y are the predictions and their associated boxes.

B.5.3 Results

To create our result, we start from a fixed configuration where we accept a box if it has a confidence score bigger then 0.01, a minimum IoU of 0.5.

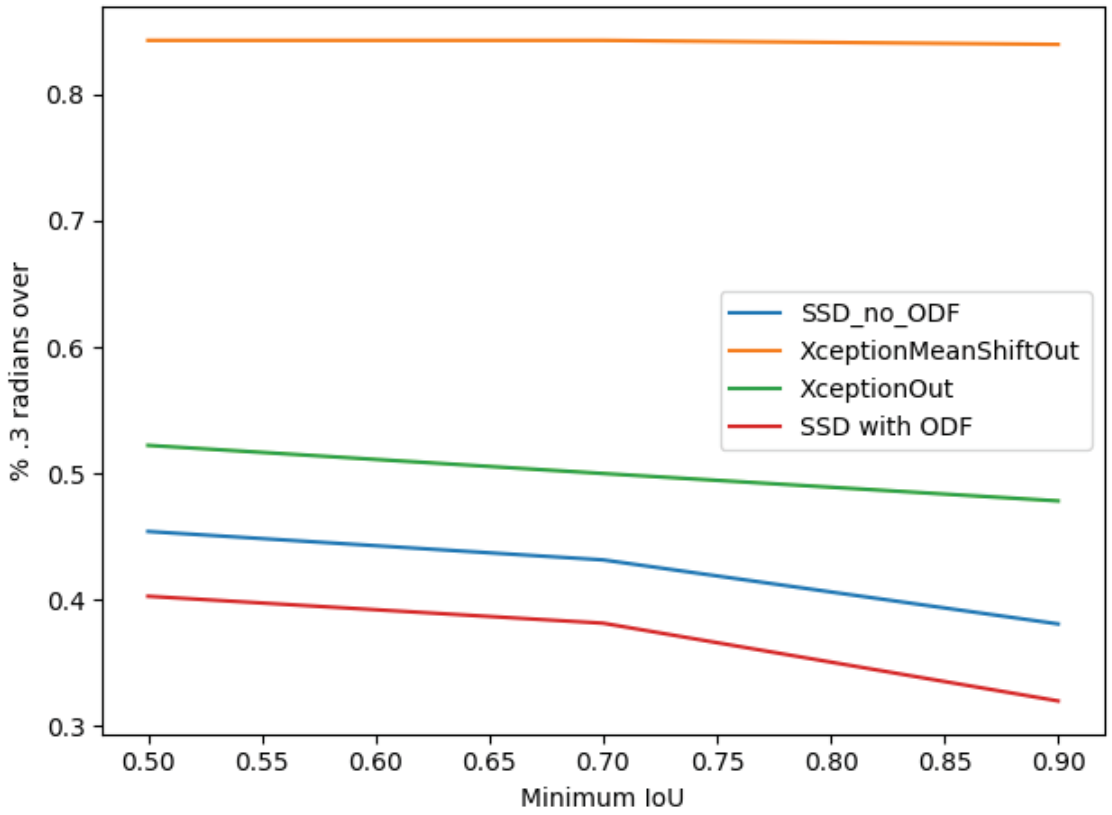


figure B.3 – Error rate by IoUs

We first analyze the effect of the box quality on the orientation estimation. For all methods but the SSD-MeanShift, the IoU has a noticeable impact on the quality of the estimation. The difference between a correct box (.5 IoU) and a perfect box (.9 IoU) can help the orientation estimation by a factor of 10%.

We observe a similar effect when analyzing the confidence of the boxes. The higher the confidence, the better the estimation.

Next, we look at the size of the box. We make the hypothesis that the orientation estimation is harder on smaller boxes, which is the case for classification and localization. We used the same approach as COCOciteand discretize the boxes in three categories based on their size : small, medium and large. In Table B.5, we present our result. Surprisingly, the *medium* vehicles are the easiest for all models and more

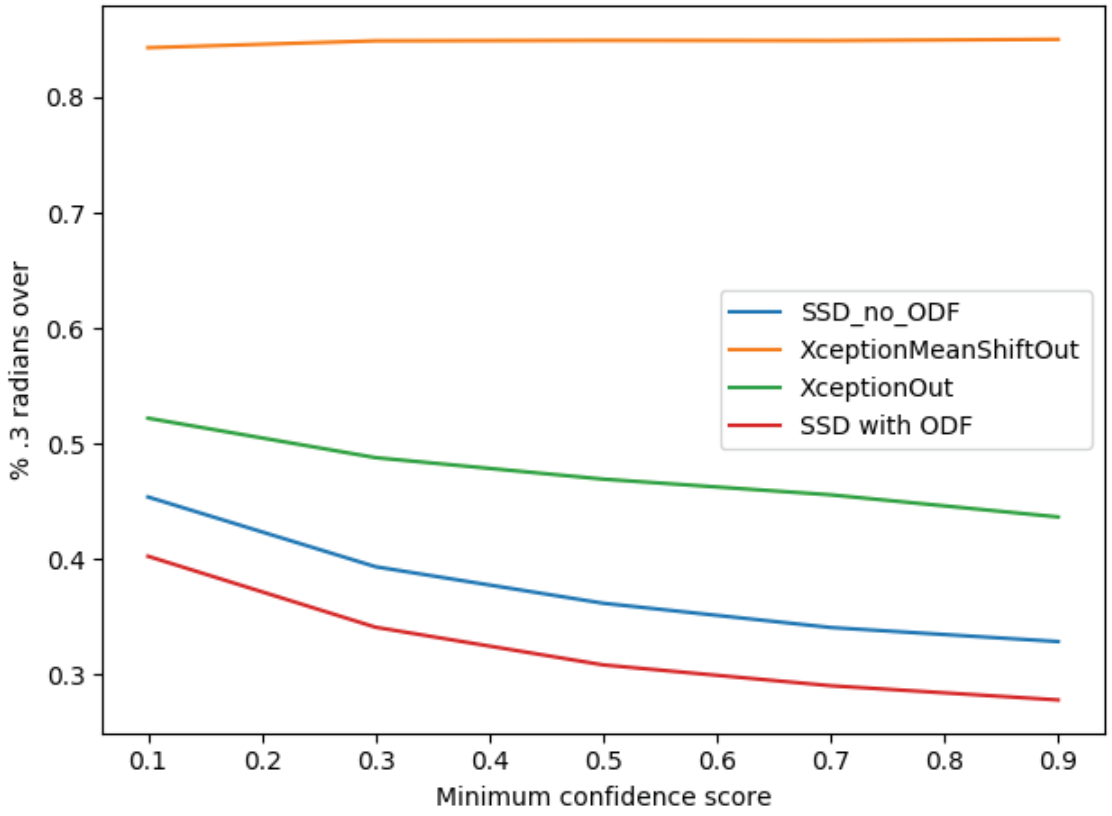


figure B.4 – Error rate by confidence

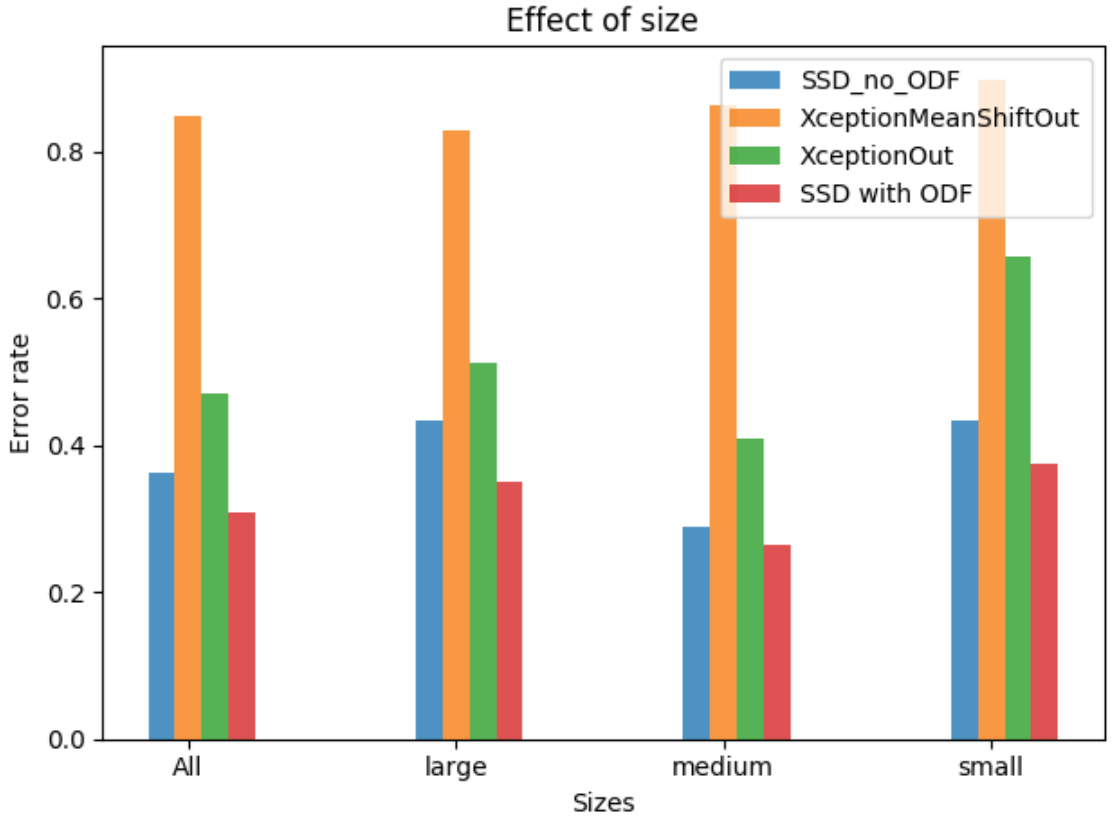


figure B.5 – Error rate by size

surprisingly, the ODF makes little difference in this case. This may be because the dataset has more *medium* vehicles than any other size.

Finally we test our method on different classes. We present the error rate associated with each class in Fig. B.6. The *pedestrian* class has the biggest error rate. It is in relation with their localization score which is also low. Pedestrians are often difficult to see which makes the prediction harder.

B.5.4 Idle analysis

Our novel output on idle classification achieves surprising scores. We present the precision-recall curves for both models in Fig. B.6. Both models achieves impressive performance, but the model with ODF is superior by a large margin.

figure B.6 – Error rate by class.

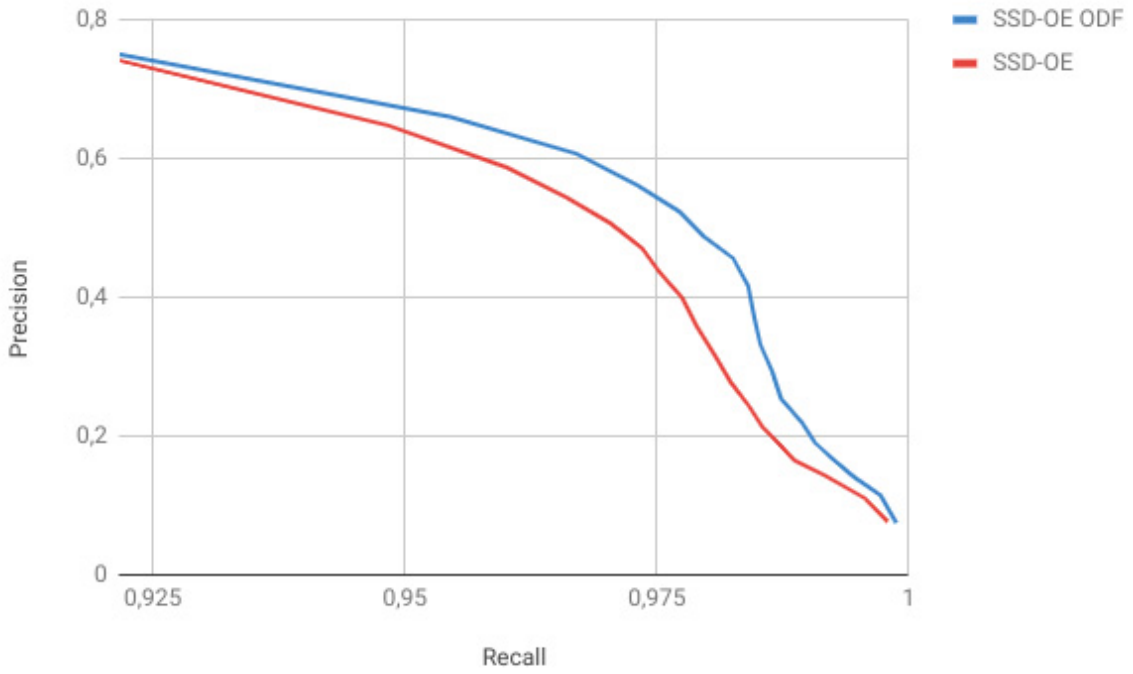


figure B.7 – Precision-Recall curves for our model with and without ODF.

B.5.5 Examples and failure cases

In Fig. B.8, we present some predictions of our model.

B.6 Future work

One main issue with our model is the cold-start problem. We do need to compute the optical flow for some time before running the model frame by frame. We could solve this problem by adding a new component to our model which would estimate the prior using recurrent networks and time series theory. While our model is quite fast, we are still 10% slower than the original SSD. We could make our model faster by using shared features and depth-wise convolutions.

figure B.8 – Examples of our model. Red arrow means that the car is estimated as parked. The title of each box represents the class and the confidence.

B.7 Conclusion

In this work, we presented a novel application of deep learning on orientation and idling estimation. We achieved a level of performance usable by an application while keeping a quick inference time.

Bibliographie

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard et others.
« TensorFlow : A System for Large-Scale Machine Learning. ».
Dans *OSDI*, volume 16, pages 265–283, 2016.

[2] Federal Highway administration.
« Guide, Traffic Monitoring ».
Dans *US Department of Transportation*, 2016.

[3] Nafees Anwar, Geoff Jones et Siva Ganesh.
« Measurement of data complexity for classification problems with unbalanced data ».
Dans *Statistical Analysis and Data Mining : The ASA Data Science Journal*, volume 7, pages 194–211. Wiley Online Library, 2014.

[4] Saeid Asgari Taghanaki, Aïcha BenTaieb, Anmol Sharma, Shaohua Kevin Zhou, Yefeng Zheng, Bogdan Georgescu, Puneet Sharma, Sasa Grbic, Zhoubing Xu, Dorin Comaniciu et Ghassan Hamarneh.
« Select, Attend, and Transfer : Light, Learnable Skip Connections ».
Dans *ArXiv*, volume abs/1804.05181, 2018.

[5] Muhammet Bastan, Kim-Hui Yap et Lap-Pui Chau.
« Remote Detection of Idling Cars Using Infrared Imaging and Deep Networks ».
Dans *arXiv preprint arXiv :1804.10805*, 2018.

[6] Richard Baumgartner et Ray L Somorjai.
« Data complexity assessment in undersampled classification of high-

dimensional biomedical data ».
Dans *Pattern Recognition Letters*, volume 27, pages 1383–1389. Elsevier, 2006.

[7] Dan Becker.
« Hot Dog - Not Hot Dog », 2017.
[En ligne]. Available : https://www.kaggle.com/dansbecker/hot-dog-not-hot-dog.

[8] Ingwer Borg et P Groenen.
Modern multidimensional scaling : theory and applications.
volume 40. Wiley Online Library, 2003.

[9] Lukas Bossard, Matthieu Guillaumin et Luc Van Gool.
« Food-101 – Mining Discriminative Components with Random Forests ».
Dans *Proc. ECCV*, pages 446–461, 2014.

[10] André L Brun, Alceu S Britto Jr, Luiz S Oliveira, Fabricio Enembreck et Robert Sabourin.
« A framework for dynamic classifier selection oriented by the classification problem difficulty ».
Dans *Pattern Recognition*, volume 76, pages 175–190. Elsevier, 2018.

[11] Y. Bulatov.
« notMNIST », 2017.
[En ligne]. Available : www.kaggle.com/lubaroli/notmnist.

[12] Shulian Cai, Jiabin Huang, Delu Zeng, Xinghao Ding et John Paisley.
« MEnet : A Metric Expression Network for Salient Object Segmentation ».
Dans *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 598–605. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[13] François Chollet et others.
« Keras », 2015.
[En ligne] Available :https://github.com/fchollet/keras.

[14] François Chollet.
« Xception : Deep Learning with Depthwise Separable Convolutions ».
Dans *Proc. CVPR*, pages 1800–1807, 2017.

[15] Adam Coates, Andrew Ng et Honglak Lee.
« An analysis of single-layer networks in unsupervised feature learning ».
Dans *Proc. AIStat*, pages 215–223, 2011.

[16] J. Cohen.
« A coefficient of agreement for nominal scales ».
Dans *Educ. Psychol. Meas.*, volume 20, pages 37–46, 1960.

[17] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth et Bernt Schiele.
« The cityscapes dataset for semantic urban scene understanding ».
Dans *Proc.CVPR*, pages 3213–3223, 2016.

[18] Lisa Cummins et Derek Bridge.
« Choosing a case base maintenance algorithm using a meta-case base ».
Dans *Research and Development in Intelligent Systems XXVIII*, pages 167–180. Springer, 2011.

[19] Jonathan D. Regehr et Rob Poapst.
« *A new guide for traffic monitoring practitioners and traffic data customers in Canada* », 2017 (accessed June, 2018).

[20] Jifeng Dai, Yi Li, Kaiming He et Jian Sun.
« R-FCN : Object detection via region-based fully convolutional networks ».
Dans *Proc. NIPS*, pages 379–387, 2016.

[21] Navneet Dalal et Bill Triggs.
« Histograms of oriented gradients for human detection ».
Dans *Proc. CVPR*, pages 886–893, 2005.

[22] Navneet Dalal et Bill Triggs.
« INRIA person dataset ».
Dans *[En ligne]. Available : http ://pascal. inrialpes. fr/data/human*, 2005.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li et L. Fei-Fei.
« ImageNet : A Large-Scale Hierarchical Image Database ».
Dans *Proc. CVPR*, 2009.

[24] Piotr Dollar, Christian Wojek, Bernt Schiele et Pietro Perona.
« Pedestrian detection : An evaluation of the state of the art ».
Dans *IEEE Trans. Pattern Anal. Mach. Intell.*, volume 34, pages 743–761, 2012.

[25] Zhen Dong, Yuwei Wu, Mingtao Pei et Yunde Jia.
« Vehicle type classification using a semisupervised convolutional neural network ».
Dans *IEEE Trans. Intell. Transp. Syst.*, volume 16, pages 2247–2256, 2015.

[26] Robert PW Duin et Elżzbieta Pękalska.
« Object representation, sample size, and data set complexity ».
Dans *Data complexity in pattern recognition*, pages 25–58. Springer, 2006.

[27] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn et A. Zisserman.
« The Pascal Visual Object Classes (VOC) Challenge ».
Dans *International Journal of Computer Vision*, volume 88, pages 303–338, juin 2010.

[28] Luís PF Garcia, André CPLF de Carvalho et Ana C Lorena.
« Effect of label noise in the complexity of classification problems ».
Dans *Neurocomputing*, volume 160, pages 108–119. Elsevier, 2015.

[29] Luís PF Garcia, André CPLF de Carvalho et Ana C Lorena.
« Noise detection in the meta-learning level ».
Dans *Neurocomputing*, volume 176, pages 14–25. Elsevier, 2016.

[30] Andreas Geiger, Philip Lenz et Raquel Urtasun.
« Are we ready for autonomous driving ? the kitti vision benchmark suite ».
Dans *Proc. CVPR*, pages 3354–3361, 2012.

[31] Ross Girshick.
« Fast r-cnn ».
Dans *Proc. ICCV*, pages 1440–1448, 2015.

[32] Ross Girshick, Jeff Donahue, Trevor Darrell et Jitendra Malik.
« Rich feature hierarchies for accurate object detection and semantic segmentation ».
Dans *Proc. CVPR*, pages 580–587, 2014.

[33] Ian Goodfellow, Yoshua Bengio et Aaron Courville.
Deep Learning.
MIT Press, 2016.
http://www.deeplearningbook.org.

[34] Michael Greenacre et R Primicerio.
« Measures of distance between samples : non-Euclidean ».
Dans *Multivariate analysis of ecological data*, pages 5–1, 2013.

[35] Ricardo Guerrero-Gómez-Olmedo, Roberto J López-Sastre, Saturnino Maldonado-Bascón et Antonio Fernández-Caballero.
« Vehicle tracking by simultaneous detection and viewpoint estimation ».
Dans *Proc. WCIBNAC*, pages 306–316, 2013.

[36] Ricardo Guerrero-Gómez-Olmedo, Beatriz Torre-Jiménez, Roberto López-Sastre, Saturnino Maldonado-Bascón et Daniel Oñoro-Rubio.
« Extremely overlapping vehicle counting ».
Dans *in proc of ICPRIA*, pages 423–431, 2015.

[37] Jinjiang Guo, Pengyuan Ren, Aiguo Gu, Jian Xu et Weixin Wu.
« Locally Adaptive Learning Loss for Semantic Image Segmentation ».
Dans *CoRR*, volume abs/1802.08290, 2018.

[38] K. Hara, R. Vemulapalli et R. Chellappa.
« Designing Deep Convolutional Neural Networks for Continuous Object Orientation Estimation ».
Dans *ArXiv e-prints*, février 2017.

[39] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P-M. Jodoin et H. Larochelle.
« Brain Tumor Segmentation with Deep Neural Networks ».
Dans *Med. Image Anal.*, volume 35, pages 18–31, 2017.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun.
« Deep Residual Learning for Image Recognition ».
Dans *Proc. CVPR*, pages 770–778, 2016.

[41] Tin Kam Ho et Mitra Basu.
« Complexity measures of supervised classification problems ».
Dans *IEEE transactions on PAMI*, volume 24, pages 289–300. IEEE, 2002.

[42] Derek Hoiem, Yodsawalai Chodpathumwan et Qieyun Dai.
« Diagnosing error in object detectors ».
Dans *Proc. ECCV*. Springer, 2012.

[43] Jie Hu, Li Shen et Gang Sun.
« Squeeze-and-Excitation Networks ».
Dans *Proc. CVPR*, 2018.

[44] Gao Huang, Zhuang Liu, Kilian Q Weinberger et Laurens van der Maaten.
« Densely connected convolutional networks ».
Dans *Proc. CVPR*, pages 2261–2269, 2017.

[45] Sergey Ioffe et Christian Szegedy.
« Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift ».
Dans *Proc. ICML*, volume 15.

[46] Stefan Jaeger, Sema Candemir, Sameer Antani, Yì-Xiáng J Wáng, Pu-Xuan Lu et George Thoma.
« Two public chest X-ray datasets for computer-aided screening of pulmonary diseases ».
Dans *Quantitative imaging in medicine and surgery*, volume 4, page 475. AME Publications, 2014.

[47] Tony Jebara, Risi Kondor et Andrew Howard.
« Probability product kernels ».
Dans *JMLR*, volume 5, pages 819–844, 2004.

[48] Heechul Jung, Min-Kook Choi, Jihun Jung, Jin-Hee Lee, Soon Kwon et Woo Young Jung.
« ResNet-Based Vehicle Classification and Localization in Traffic Surveillance Systems ».
Dans *Proc. CVPRW*, pages 934–940, 2017.

[49] V Kastrinaki, Michalis Zervakis et Kostas Kalaitzakis.
« A survey of video processing techniques for traffic applications ».
Dans *Image vision comput.*, volume 21, pages 359–381, 2003.

[50] Pyong-Kun Kim et Kil-Taek Lim.
« Vehicle Type Classification Using Bagging and Convolutional Neural Network on Multi View Surveillance Image ».
Dans *Proc. CVPRW*, pages 914–919, 2017.

[51] D. Kingma et J. Ba.
« Adam : A Method for Stochastic Optimization ».
Dans *Proc. ICLR*, 2015.

[52] Jonathan Krause, Michael Stark, Jia Deng et Li Fei-Fei.
« 3d object representations for fine-grained categorization ».
Dans *Proc. CVPRW*, pages 554–561, 2013.

[53] A. Krizhevsky et G. Hinton.
« Learning multiple layers of features from tiny images ».
Dans *Tech.Report*, 2009.

[54] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton.
« Imagenet classification with deep convolutional neural networks ».
Dans *Proc. NIPS*, pages 1097–1105, 2012.

[55] Yann LeCun, Léon Bottou, Yoshua Bengio et Patrick Haffner.
« Gradient-based learning applied to document recognition ».
Dans *Proc. IEEE*, volume 86, pages 2278–2324. IEEE, 1998.

[56] Yann Lecun, Léon Bottou, Yoshua Bengio et Patrick Haffner.
« Gradient-based learning applied to document recognition ».
Dans *Proc. IEEE*, pages 2278–2324, 1998.

[57] Yann LeCun, Corinna Cortes et CJ Burges.
« MNIST handwritten digit database ».
Dans *AT&T Labs [En ligne]. Available : http ://yann. lecun. com/exdb/mnist*, volume 2, 2010.

[58] J. T. Lee et Y. Chung.
« Deep Learning-Based Vehicle Classification Using an Ensemble of Local Ex-

pert and Global Networks ».
Dans *Proc. CVPRW*, pages 47–52, 2017.

[59] Enrique Leyva, Antonio González et Raul Perez.
« A set of complexity measures designed for applying meta-learning to instance selection ».
Dans *IEEE Transactions on Knowledge and Data Engineering*, volume 27, pages 354–367. IEEE, 2015.

[60] Chunyuan Li, Heerad Farkhoor, Rosanne Liu et Jason Yosinski.
« Measuring the Intrinsic Dimension of Objective Landscapes ».
Dans *Proc. ICLR*, 2018.

[61] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár et C Lawrence Zitnick.
« Microsoft coco : Common objects in context ».
Dans *Proc. ECCV*, pages 740–755, 2014.

[62] H. Liu, S. Chen et N. Kubota.
« Intelligent Video Systems and Analytics : A Survey ».
Dans *IEEE Trans. Ind. Informat.*, volume 9, pages 1222–1233, 2013.

[63] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu et Alexander C Berg.
« SSD : Single shot multibox detector ».
Dans *Proc. ECCV*, pages 21–37, 2016.

[64] Ana C Lorena, Luís PF Garcia, Jens Lehmann, Marcilio CP Souto et Tin K Ho.
« How Complex is your classification problem ? A survey on measuring classification complexity ».
Dans *arXiv preprint arXiv :1808.03591*, 2018.

[65] Wenhan Luo, Xiaowei Zhao et Tae-Kyun Kim.
« Multiple object tracking : A review ».
Dans *arXiv preprint arXiv :1409.7618*, volume 1, 2014.

[66] Z Luo, F B.Charron, C Lemaire, J Konrad, S Li, A Mishra, A Achkar, J Eichel et P-M Jodoin.

« MIO-TCD : A new benchmark dataset for vehicle classification and localization ».
Dans *In press at IEEE Trans. on Img. Proc.*, 2018.

[67] Zhiming Luo, Justin Eichel, Andrew Achkar, Carl Lemaire, Janusz Konrad, Akshaya Mishra, Shaozi Li, Frederic B-Charron et Pierre-Marc Jodoin.
« Traffic Surveillance Workshop and Challenge (CVPR 2017) », 2017.
http://tcd.miovision.com.

[68] Zhiming Luo, Akshaya Kumar Mishra, Andrew Achkar, Justin A. Eichel, Shaozi Li et Pierre-Marc Jodoin.
« Non-local Deep Features for Salient Object Detection ».
Dans *Proc. CVPR*, pages 6593–6601, 2017.

[69] Núria Macià, Ester Bernadó-Mansilla, Albert Orriols-Puig et Tin Kam Ho.
« Learner excellence biased by data set selection : A case for data characterisation and artificial data sets ».
Dans *Pattern Recognition*, volume 46, pages 1054–1066. Elsevier, 2013.

[70] Rafael G Mantovani, André LD Rossi, Joaquin Vanschoren, Bernd Bischl et André CPLF Carvalho.
« To tune or not to tune : recommending when to adjust SVM hyper-parameters via meta-learning ».
Dans *Proc. IJCNN*, pages 1–8. IEEE, 2015.

[71] Alina Marcu, Dragos Costea, Emil Slusanschi et Marius Leordeanu.
« A Multi-Stage Multi-Task Neural Network for Aerial Scene Interpretation and Geolocalization ».
Dans *ArXiv*, volume abs/1804.01322, 2018.

[72] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado et Jeffrey Dean.
« Distributed Representations of Words and Phrases and Their Compositionality ».
Dans *Proc. NIPS*, 2013.

[73] Dmytro Mishkin et Jiri Matas.
« All you need is a good init ».
Dans *Proc. ICLR*, 2016.

[74] Tom Mitchell.
Machine Learning.
1997.

[75] B Mohar.
« Some applications of Laplace eigenvalues of graphs ».
Dans Geňa Hahn et Gert Sabidussi, éditeurs, *Graph Symmetry : Algebraic Methods and Applications*, pages 225–275.
Springer, 1997.

[76] Gleison Morais et Ronaldo C Prati.
« Complex network measures for data set characterization ».
Dans *Proc. BRACIS*, pages 12–18. IEEE, 2013.

[77] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu et Andrew Y Ng.
« Reading digits in natural images with unsupervised feature learning ».
Dans *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011(2), page 5, 2011.

[78] Hyeonwoo Noh, Seunghoon Hong et Bohyung Han.
« Learning Deconvolution Network for Semantic Segmentation ».
Dans *Proc. ICCV*, 2015.

[79] E Nowakowska, J Koronacki et S Lipovetsky.
« Tractable Measure of Component Overlap for Gaussian Mixture Models ».
Dans *ArXiv – 1407.7172*, 2014.

[80] Lucas Chesini Okimoto, Ricardo Manhães Savii et Ana Carolina Lorena.
« Complexity Measures Effectiveness in Feature Selection ».
Dans *Intelligent Systems (BRACIS), 2017 Brazilian Conference on*, pages 91–96. IEEE, 2017.

[81] Albert Orriols-Puig, Núria Macia et Tin Kam Ho.
« Documentation for the data complexity library in C++ ».
Dans *Universitat Ramon Llull, La Salle*, volume 196, 2010.

[82] Adam Paszke et others.
« PyTorch », 2016.
https://github.com/pytorch/pytorch.

[83] Thaise M Quiterio et Ana C Lorena.
« Using complexity measures to determine the structure of directed acyclic graphs in multiclass classification ».
Dans *Applied Soft Computing*, volume 65, pages 428–442. Elsevier, 2018.

[84] Joseph Redmon, Santosh Divvala, Ross Girshick et Ali Farhadi.
« You only look once : Unified, real-time object detection ».
Dans *Proc. CVPR*, pages 779–788, 2016.

[85] Joseph Redmon et Ali Farhadi.
« YOLO9000 : Better, Faster, Stronger ».
Dans *Proc. CVPR*, 2017.

[86] Joseph Redmon et Ali Farhadi.
« YOLO9000 : Better, Faster, Stronger ».
Dans *Proc. CVPR*, 2017.

[87] Shaoqing Ren, Kaiming He, Ross Girshick et Jian Sun.
« Faster r-cnn : Towards real-time object detection with region proposal networks ».
Dans *Proc. NIPS*, pages 91–99, 2015.

[88] Raul Rojas.
« Why the normal distribution ».
Dans *Freis Universitat Berlin lecture notes*, 2010.

[89] Olaf Ronneberger, Philipp Fischer et Thomas Brox.
« U-Net : Convolutional Networks for Biomedical Image Segmentation ».
Dans Nassir Navab, Joachim Hornegger, William M. Wells et Alejandro F. Frangi, éditeurs, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241. Springer International Publishing, 2015.

[90] Frank Rosenblatt.
« The perceptron : a probabilistic model for information storage and organization in the brain. ».
Dans *Psychological review*, volume 65, page 386. American Psychological Association, 1958.

[91] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein et others.
« Imagenet large scale visual recognition challenge ».
Dans *IJCV*, volume 115, pages 211–252, 2015.

[92] José A SáEz, JuliáN Luengo et Francisco Herrera.
« Predicting noise filtering efficacy with data complexity measures for nearest neighbor classification ».
Dans *Pattern Recognition*, volume 46, pages 355–364. Elsevier, 2013.

[93] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan et Stefan Carlsson.
« CNN features off-the-shelf : an astounding baseline for recognition ».
Dans *Proc. CVPRW*, pages 806–813, 2014.

[94] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen et Xiangyang Xue.
« Dsod : Learning deeply supervised object detectors from scratch ».
Dans *Proc. ICCV*, 2017.

[95] K. Simonyan et A. Zisserman.
« Very Deep Convolutional Networks for Large-Scale Image Recognition ».
Dans *Proc. ICLR*, 2015.

[96] Sezer Sorgun et Serife Buyukkose.
« Bounds for the largest Laplacian eigenvalues of weighted graphs ».
Dans *International Journal of Combinatorics*, volume 2013, 2013.

[97] José Martínez Sotoca, Ramón Alberto Mollineda et José Salvador Sánchez.
« A meta-learning framework for pattern classication by means of data com-

plexity measures ».
Dans *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, volume 10. Asociación Española para la Inteligencia Artificial, 2006.

[98] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever et Ruslan Salakhutdinov.
« Dropout : A Simple Way to Prevent Neural Networks from Overfitting ».
Dans *J. Mach. Learn. Res.*, volume 15, pages 1929–1958, janvier 2014.

[99] Rupesh Kumar Srivastava, Klaus Greff et Jürgen Schmidhuber.
« Training Very Deep Networks ».
Dans *Proc. NIPS*, pages 2377–2385, 2015.

[100] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens et Zbigniew Wojna.
« Rethinking the inception architecture for computer vision ».
Dans *Proc. CVPR*, pages 2818–2826, 2016.

[101] Rajkumar Theagarajan, Federico Pala et Bir Bhanu.
« EDeN : Ensemble of Deep Networks for Vehicle Classification ».
Dans *Proc. CVPRW*, pages 906–913, 2017.

[102] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers et Arnold WM Smeulders.
« Selective search for object recognition ».
Dans *Proc. IJCV*, volume 104, pages 154–171, 2013.

[103] L Van Der Maaten et G Hinton.
« Visualizing high-dimensional data using t-sne. journal of machine learning research ».
Dans *J Mach Learn Res*, volume 9, page 26, 2008.

[104] Christiaan M Van der Walt et Etienne Barnard.
« Data characteristics that determine classifier performance ».
Dans *SAIEE Africa Research Journal*, volume 9, pages 87–93, 2006.

[105] U Von Luxburg.
« A tutorial on spectral clustering ».
Dans *Statistics and computing*, volume 17, pages 395–416. Springer, 2007.

[106] L. Wang, Y. Zhang et J. Feng.
« On the euclidean distance of images ».
Dans *IEEE Trans on PAMI*, volume 27, 2005.

[107] Tao Wang, Xuming He, Songzhi Su et Yin Guan.
« Efficient Scene Layout Aware Object Detection for Traffic Surveillance ».
Dans *Proc. CVPRW*, pages 53–60, 2017.

[108] B-F Wu, C-C Kao, J-H Juang et Y-S Huang.
« A New Approach to Video-Based Traffic Surveillance Using Fuzzy Hybrid Information Inference Mechanism ».
Dans *IEEE Trans. Intell. Transp. Syst.*, volume 14, pages 485–491, 2013.

[109] Linjie Yang, Ping Luo, Chen Change Loy et Xiaoou Tang.
« A large-scale car dataset for fine-grained categorization and verification ».
Dans *Proc. CVPR*, pages 3973–3981, 2015.

[110] Jason Yosinski, Jeff Clune, Yoshua Bengio et Hod Lipson.
« How transferable are features in deep neural networks ? ».
Dans *Proc. NIPS*, pages 3320–3328, 2014.

[111] Fisher Yu et Vladlen Koltun.
« Multi-Scale Context Aggregation by Dilated Convolutions ».
Dans *Proc. ICLR*, 2016.

[112] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht et Oriol Vinyals.
« Understanding deep learning requires re-thinking generalization ».
Dans *Proc. ICLR*, 2017.

[113] T. Zhang, S. Liu, C. Xu et H. Lu.
« Mining Semantic Context Information for Intelligent Video Surveillance of Traffic Scenes ».
Dans *IEEE Trans. Ind. Informat.*, volume 9, pages 149–160, 2013.

[114] Zhe Zhu, Dun Liang, Songhai Zhang, Xiaolei Huang, Baoli Li et Shimin Hu.
« Traffic-sign detection and classification in the wild ».
Dans *Proc. CVPR*, pages 2110–2118, 2016.